
# PostScript Printer Description File Format Specification

*Adobe Developer Support*

Version 4.1

9 April 1993

Adobe Systems Incorporated

Corporate Headquarters
1585 Charleston Road PO Box 7900
Mountain View, CA 94039-7900
(415) 961-4400 Main Number
(415) 961-4111 Developer Support
Fax: (415) 961-3769

Adobe Systems Europe B.V.
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z-O, Netherlands
+31-20-6511 200
Fax: +31-20-6511 300

Adobe Systems Eastern Region
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120
Fax: (617) 273-2336

Adobe Systems Japan
Swiss Bank House 7F
4-1-8 Toranomon, Minato-ku
Tokyo 105, Japan
+81-3-3437-8950
Fax: +81-3-3437-8968

PN LPS5003

# Contents

# PostScript Printer Description File Format Specification

## 1 Introduction

PostScript™ printer description files (also known as PPD files) are human-readable, machine-parsable text files that provide a uniform approach to using the diverse special features of devices that contain PostScript interpreters. These features include different page sizes, different methods of paper and film handling, memory size, font availability, and finishing features such as duplex printing and stapling. All devices do not have the same set of features, and even devices with the same features do not necessarily invoke those features in the same way.

The information contained in PPD files serves as a list of available features, as basis for building a user interface, and as a mechanism for invoking the features on a particular device.

In this specification, the term device means any output device containing a PostScript interpreter, such as a printer, imagesetter, or film recorder. Each device has a PPD file associated with it. The PPD files for all devices that are accessible to a given host computer are stored on that host computer, where they can be parsed by applications.

Applications can parse PPD files to discover the list of available features on the currently selected device. PPD files contain structures that allow “blind” parsing of a list of features. Applications can parse for these structures without understanding the features they contain. Applications can then build a user interface from the list of features found in the PPD file for the selected device.

The PPD file also contains the PostScript language code to invoke each feature. In this specification, the term output file refers to the file containing the PostScript language description of the document composed by the user. When a user selects a feature from the user interface, such as manual feed or duplex printing, the code for each selected feature is extracted from the PPD file and included in the output file before the output file is sent to the device.

Local customizations to a PPD file can be added at the user site to accommodate additions to the printer, such as the addition of fonts or memory, or to configure a device a certain way (for example, to always print on both sides of the paper).

There is a close relationship between PPD files and the Adobe Systems *document structuring conventions* (also known as DSC). These comment conventions can be used in an output file to identify the code that invokes device-specific features. This allows the output file to be redirected from one device to another by a spooler or other post-processing software. As an output file is routed across a network, a spooler can extract device-specific code by parsing for the associated DSC comments. The spooler can then parse the PPD file for the new device, extract new device-specific code, and insert new device-specific code to satisfy the needs of the output file.

Every piece of code that is extracted from a PPD file and inserted into an output file should be enclosed by the appropriate DSC comments. Version 3.0 of the Document Structuring Conventions specification is documented in Appendix G of the *PostScript Language Reference Manual, Second Edition*. Any later versions of this specification can be obtained from the Adobe™ Developers Association.

## 2 Using PPD Files

PPD files can be used during several phases of document production. First, the user selects an output device and, implicitly or explicitly, a PPD file. The association of the PPD file with the printer can be handled by an application, or the user may select the PPD file explicitly. At this time, the PPD file can be used as data for a configuration tool An application can parse the PPD file for a list of the optional accessories, and display a configuration panel that asks the user which accessories are installed. This information can be used later by a printing application to choose which options to display to the user at print time.

*Note* *This specification does not address the uses of PPD files at document composition time. For information about using a PPD file at document composition time, see Technical Note #5117, "Supporting Device Features."*

At print time, the selected PPD file can be used to construct a user interface that displays the available features of the requested device, such as duplex printing or manual feeding. After the user selects various printing features, the code to invoke those features can be extracted from the PPD file and inserted into the output file. Finally, PPD files for other devices can be used by a post-processor, such as a spooler, to insert new device-specific code into the output file and route the file to a different device. This section provides more detail on the use of PPD files in each phase of document production.

In this specification, the application that parses the PPD file for device features and provides the print panel function is referred to as a print manager. Often, it is the same piece of software that converts an application's internal representation of a document to the PostScript language representation of the same document. The function of the print manager might be provided by a system-level driver, by a separate piece of software, or it might be part of an application.

Among its other duties, the print manager

- takes input from the user via some user interface, such as a print panel or command line,
- extracts from the PPD file the corresponding code sequences to invoke the requested features,
- inserts the code sequences into the appropriate setup section of the output file, and
- surrounds the code sequences with the appropriate DSC comments.

Some device features require additional memory or other hardware before they can be invoked. For example, a device might need more than the minimum amount of memory to print a legal-size page, or to do color separations, or it might need an external device attached to fold paper. The PPD file for the device will record support for all of these features if they are supported by the device hardware and the PostScript interpreter; it is up to the user to install the correct peripherals and memory needed to make these features accessible.

## 2.1 Building a User Interface for Printing

At print time, a user must be able to select various device features, such as paper size or manual feed, through a user interface such as a print panel or a command line. The features offered to the user by a print panel can be constructed by parsing the PPD file for the selected device, discovering the available features, and displaying them to the user for selection. For example, the PPD file contains a list of paper sizes supported by the device. A user interface can display that list to the user and allow the user to select a paper size from the list.

The PPD file also contains information about the default state of the device as it is shipped from the factory. The default state of the device can be used as a starting point for setting the initial state of the user interface. For example, the default state of optional accessories can be used to indicate whether or not those accessories are installed, and, therefore, whether or not to display them to the user.

Second, the default state of individual features can be used to determine how they are initially displayed. For example, if the default state of the device is to print on letter-size paper with manual feed turned off, the user interface could initially appear with letter-size paper selected and manual feed not selected. This tells the users that if they change nothing, their documents will be printed on letter-size paper with manual feed disabled. The PPD file can thus be used both to tell users both what they can do and what will happen if they do nothing.

It is important to realize that the defaults do not necessarily reflect the current state of the device, as any previous job could have changed the state of the device. Print managers should ensure that if the user selects nothing else, the defaults shown in the user interface are invoked.

## 2.2 Inserting Print-Time Features

When the user has finished selecting features, the print manager can consult the PPD file for additional information, such as

- whether this is a Level 1 or a Level 2 device, so the print manager knows whether or not it can generate code that uses Level 2 features
- if it is a Level 1 device, which extensions to the PostScript language are supported, if any
- the code sequences that invoke the features the user has selected via the user interface
- any additional information that the author of the print manager thinks would be useful in generating an efficient output file.

Armed with information, the print manager converts the internal representation of the document into the PostScript language representation of the document. It includes the device-specific code for the features requested by the user, and surrounds these feature requests with DSC comments for possible later parsing by other applications.

The following example shows a PostScript language output file that describes a very small document. In this example, the output file does not yet contain DSC comments or device-specific code. Throughout this section, this output file will grow as DSC comments and device-specific code are added.

```
/sp /showpage load def
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
```

In the next example, assume that the user requested letter-size paper via some user interface. The print manager extracts from the PPD file the device-specific code to invoke letter-size paper, inserts the code into the output file, inserts the appropriate DSC comments, and sends the output file to the output device.

The following is the example with DSC comments and device-specific code added. The code sequence extracted from a PPD file is in boldface.

```
%!PS-Adobe-3.0
%%Title: test.ps
%%EndComments
/sp /showpage load def
%%EndProlog
%%BeginSetup
%%BeginFeature: *PageSize Letter
   statusdict /lettertray get exec
%%EndFeature
%%EndSetup
%%Page: one 1
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
%%Trailer
```

When the output file is sent to the output device, the interpreter ignores the comments and executes the PostScript language commands, including the code sequence that sets up the letter-size input tray.

For most user-selectable features of a device, there is no clear inverse operation. That is, unsetting, for example, a ledger-size paper tray will typically mean establishing a different paper tray as the current paper tray. Explicitly setting the device back to its default condition has the same effect; it will “undo” the effects of having previously set a given feature. Unless there is a specific reason to do so, it is not necessary to reverse the effects of invoking device-specific features for any particular print job, since the job server should provide that service, returning device features to their default settings at the end of each job.

## 2.3 Post-Processing

In some environments, there might be a post-processor, such as a spooler, which also acts as a print manager. In this context, the requested device might be unavailable, and the print manager/spooler might need to redirect an output file from one device to another. If an output file is to be redirected, the print manager parses the DSC comments in the output file, and strips out the original device-specific code. It then parses the PPD file of the newly selected

device, extracts from the new PPD file the device-specific code requested by the DSC comments, inserts the device-specific code from the new PPD file into the output file, and sends the output file to the new device.

The following is the example file as it is sent to the new device (note that the device-specific code is different):

```
%!PS-Adobe-3.0
%%Title: test.ps
%%LanguageLevel: "2"
%%EndComments
/sp /showpage load def
%%EndProlog
%%BeginSetup
%%BeginFeature: *PageSize Letter
   1 dict dup /PageSize [612 792] put setpagedevice
%%EndFeature
%%EndSetup
%%Page: one 1
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
%%Trailer
```

## 2.4 Error Handling

It is possible that a print manager might encounter PPD files that contain errors. Print managers are encouraged to include a reasonable level of error-handling. Examples of possible errors that need handling are dangling symbolic references (section 5.6), missing information about page sizes (section 5.8, section 5.10, and section 5.11), and missing required keywords (section 3.4).

Files that conform to this specification are portable across systems and communication channels, with the exception of system-dependent filenames in *Include statements (see section 2.6 and section 4.2.) Non-conforming files might be accepted by some software, but their portability is at risk. For example, some software might accept out-of-range byte codes in a PPD file and treat them in a system-dependent manner, but when ported to another system with stricter software, the same file might fail.

When inserting invocation code from a PPD file into a job stream, print managers are encouraged to execute such code in a **stopped** context to catch any errors, and to surround the code with **mark** and **cleartomark** to ensure that the operand stack is cleaned up if an error occurs while executing the code.

## 2.5 Order Dependencies

When a print manager is inserting device-specific code into an output file, the order of certain operators with respect to each other is important and must be considered. The following are the types of operators concerned, shown in their respective order

- resolution invocations
- media tray invocations
- media size/imageable region invocations

Any of the types of operators listed can occur in either the document setup or page setup sections of the output file. For information about the document setup and page setup sections of an output file, see the Document Structuring Conventions specification referenced on page 2.

If any of these types of invocations occur together in any setup section, the order shown above must be maintained to achieve the expected results. Specifically, the following guidelines for ordering should be applied:

- Any resolution invocation (available only on devices where the user can change the resolution of the device via software) must occur before any media tray or media size selection. This is important because on many devices, the resolution is not actually set until the tray or size selection occurs, so the tray or size selection must occur after the resolution invocation.
- If both a specific media tray invocation (for example, Upper) and a specific imageable region invocation (for example, Letter) occur in the output file, the media tray invocation should precede the imageable region invocation. Otherwise, the tray invocation might invalidate the imageable region invocation.

The following items generally occur after media size selection:

- job control requests such as duplex, automatic tray switching, signaturing, output bin selection, fine tuning, facsimile, and finishing features
- halftone invocations (including halftone screen setup, transfer functions, and accurate screens)

Any modifications to the halftone screens, including modifications to the screen angle, screen frequency, spot function, transfer function, and the invocation of the accurate screens feature, must occur after the media size invocation. This is because the media size invocation will set the halftone screen

settings to their default settings. Modifications to the halftone screens are not confined to the setup sections; they can occur anywhere on a page in the output file.

The keyword *OrderDependency provides information about the appropriate setup section and ordering of each PostScript invocation in a PPD file.

## 2.6 Local Customization of PPD Files

A PPD file is a static representation of the features available on a device. It contains information on the features available on a device as it is shipped from the factory. In general, this will be the minimum amount of memory available for that device, the minimum font set, and the maximum list of optional accessories, such as paper trays, so that all the necessary invocation code is present in the PPD file, even if the accessories are not installed when shipped from the factory. Optional accessories will be marked as optional in the PPD file.

Once a device is installed, features such as additional memory and fonts might be added to the device. In this specification, the term“system administrator” is used to mean the person who adds memory and fonts and otherwise is generally responsible for the device. In a single-user, single-printer environment, the role of system administrator is typically played by the user.

The task of managing a device is a dynamic issue that requires keeping track of fonts downloaded to disk, error handlers, RAM-based fonts and procedure sets, default device setup, and so forth. This kind of device management is beyond the scope of PPD files. However, there are provisions for customizing the information contained in PPD files to adapt them to local instances of devices.

One approach to customization is for a print manager to parse all of the PPD files available on a host system and store the data into a database. The print manager (or other utility) can then update the database dynamically to accommodate additional memory, fonts, available trays, and other changeable printer features.

Another approach is to permit *editable customization files*, which include a reference to the original PPD file. In any given computing environment, there should be a single PPD file for each type of device in use. If particular software products (or individual users) want to add to or modify the contents of a PPD file, they should do so by creating a local customization file that contains entries only for items they want to change or add. For example, a customization file might contain information about additional memory and fonts that were added to the device after it was received from the factory. Using local customization files permits a system administrator to replace the original PPD without forcing users to reedit their local customization files.

To be understood by applications parsing PPD files, this local customization file must conform to the PPD specification, so in a sense, the customization file is itself a PPD file. The customization file should be given a unique name that represents a particular device (for example, *MyPrntr.PPD*). The *.PPD* extension should be preserved, with case irrelevant, in case applications or print managers are searching for files with that extension. In addition, certain application developers might create customization files with different extensions, which are read only by their application.

The local customization file and the original PPD file are related through the use of the following entry:

```
*Include: "filename"
```

Because the original PPD file should not be modified by the addition of the *Include keyword, this keyword appears in the customization file, and references the original PPD file by naming it with *filename* (see the sample local customization file below).

When a PPD file is included by another file, the parsing details change somewhat. In particular, there might be several instances of the same keyword in the "composite" file. In this case, the *first* instance of a given keyword (or, if the keyword takes an option, of a keyword-option pair) is correct. This enables a parser to ignore subsequent versions of the same entry, possibly reducing the parsing time.

*Note* *The concept of "first instance is correct" does not apply to optionless keywords such as* *PrinterError*, *which normally have multiple instances in a PPD file, and which do not have option keywords to distinguish those instances. In this case, all instances must always be parsed and recorded.*

*For a keyword such as* *FreeVM, *which would not normally have multiple entries, or for a main keyword-option keyword combination such as* *PageSize Letter, *which has an option keyword to distinguish the entry and, therefore, would normally have only one instance in a PPD file, the first instance encountered is considered correct. This implies that a parser must either have knowledge of the semantics of the PPD keywords when parsing, or it must save all instances in some form for a later, smarter processor to decide which are rightfully multiple instances. See Appendix B for a list of optionless keywords that might occur multiple times in a file.*

By logical extension, if the first instance of a keyword is the correct instance, than all keywords in a local customization file should occur *before* the *Include statement that references the original PPD file.For example, assuming the original PPD file is called *TIm35521.PPD*, a local customization file would look like this

```
*% Local Customization File for TI microLaser in Bldg A
*NickName: "microLaser - Building A"
*FreeVM: "1907408"
*DefaultManualFeed: True
*Include: "TIm35521.PPD"

*% end of local customization file
```

The local customization file might be named *Micro-A.PPD*. A parser reading this file would record the values of *NickName, *FreeVM, and *DefaultManualFeed as shown above, and would ignore subsequent occurrences of those keywords in the included PPD file, *TIm35521.PPD*.

This inclusion scheme permits the original file, *TIm35521.PPD*, to be easily replaced if a new version is issued. Users will not have to edit their local customization files to take advantage of a new version of a PPD file; if the new PPD file has the same name as the old one, it will automatically be referenced by the local customization file.

If a keyword that is normally enclosed by the *OpenUI/*CloseUI keyword pair in the original PPD file is repeated (replaced) in a local customization file, it should also be bracketed by *OpenUI/*CloseUI in the customization file. It should also include any *OpenGroup/*CloseGroup keywords, if appropriate. See section 5.5 for details on the *OpenUI/*CloseUI and *OpenGroup/*CloseGroup keywords. This means that a print manager, when parsing the PPD file, must be prepared to find multiple instances of a given *OpenUI/*CloseUI group, both in the original PPD file and in any local customization files that might exist.

### Using and Changing Default Settings

When building a user interface from a PPD file, a print manager can use the *Default- keywords to select defaults for the various features displayed to the user. For example, if the PPD file for the selected device contains this entry:

```
*DefaultManualFeed: False
```

then the print manager can indicate in the user interface that manual feeding of the media is, by default, turned off, and provide a way for the user to turn on manual feeding.

The defaults listed in the original PPD file reflect the state of the device when it is shipped from the factory. If the system administrator wants to set up the device differently, the new defaults should be included in the local customization files. For example, if the device in the previous example was set up to always feed from the manual feed slot, then the local customization file should contain this entry

```
*DefaultManualFeed: True
```

This allows the print manager to indicate in the user interface that manual feeding of the media on this device is, by default, turned on.

As noted earlier, the print manager should ensure that the code to invoke defaults is included in the output file, if no non-default option has been selected by the user.

# 3 Format

PPD files provide several fundamental kinds of information about device features, including the feature options, the default settings, how to change the settings, and other information that might be used for scheduling jobs. The syntax of PPD files is a simple line-oriented format where the options, defaults, and invocation strings (PostScript language code sequences that change a feature setting) are made available through a regular set of keywords. Where applicable, there is also information supplied about how to determine the current setting of any of these configurable options (known as querying the device ).

## 3.1 ASCII Code Chart

Throughout the next few sections, certain ASCII characters are referenced repeatedly. For completeness, they are all shown here with their decimal ASCII equivalents:

- *space* (decimal ASCII 32)
- *tab* (horizontal), (decimal ASCII 9)
- *asterisk*, ‘*’ (decimal ASCII 42)
- *colon*, ‘:’ (decimal ASCII 58)
- *slash*, ‘/’ (decimal ASCII 47)
- *question mark*, ‘?’ (decimal ASCII 63)
- *double quote*, ‘"‘ (decimal ASCII 34)

- *newline* — any combination of carriage return (decimal ASCII 13) and line feed (decimal ASCII 10)
- *caret*, '^' (decimal ASCII 94)
- *open angle bracket*, '<' (decimal ASCII 60)
- *closing angle bracket*, '>' (decimal ASCII 62)

## 3.2 Definition of Terms

This section defines many of the terms used throughout this specification.

There are two basic types of keywords: *main keywords* and *option keywords*. Main keywords describe a class of features, such as page sizes and input slots (*PageSize, *InputSlot). Main keywords either provide information about a feature, such as how much memory is available or which fonts are resident, or they provide the code to invoke a user-selectable feature, such as an input tray or manual feed.

Option keywords, which modify main keywords, describe the list of available options for a feature. For example, the option keywords for the keyword *PageSize describe the available page sizes, such as Letter, Legal, A4, Tabloid, and so on. The option keywords for the main keyword *InputSlot describe the available input slots, such as Upper, Lower, and so on.

The registry of main keywords is controlled by Adobe Systems, and all valid main keywords are described in this specification. Adobe Systems will issue an addendum to this specification when new main keywords are added to the registry.

The registry of option keywords is also controlled by Adobe Systems, but is subject to more frequent change and additions. Any OEM can add the option keywords necessary to describe new device features, but these option keywords must be approved by Adobe Systems. This is to avoid proliferation of different option keywords that mean the same thing or of one option keyword meaning different things on different devices. All currently registered option keywords are documented herein, but more can be added at any time.

*Note* *Releases of the keyword registry are uniquely identified by the date of the release. Send requests to the appropriate address on the title page of this specification.*

Two subsets of the main keyword class are *default keywords* and *query keywords*. Default keywords provide information about the default state of the device as shipped from the factory. Query keywords provide a code sequence, which, when downloaded to the device, returns information about the current state of the device. This can be useful for print managers/spoolers to deter-

mine the state of a device and perhaps request operator intervention (for example, if the appropriate media tray is not present). Queries can only be used when the physical interface to the device permits feedback from the device.

An *entry* is a convenient term used to describe a main keyword and its associated option and value, or a group of main keywords, options, and values that are specified as belonging together.

There are seven canonical forms of entries:

- *MainKey
- *MainKey: StringValue
- *MainKey: "QuotedValue"
- *MainKey: ^SymbolValue
- *MainKey OptionKey: StringValue
- *MainKey OptionKey: "InvocationValue"
- *MainKey OptionKey: ^SymbolValue

Each entry in a PPD file falls into one of these forms. The various value types are defined in section 3.6.

The line length of any line in a PPD file must be less than or equal to 255 characters, including line termination characters. Line termination in PPD files can consist of any combination of carriage return (decimal ASCII 13) and line feed (decimal ASCII 10). In this specification, the set of line termination characters is referred to as *newline*. Different computer systems can represent the line endings differently without affecting the actual data representation.

*White space* is defined as any combination of the characters *space* and *tab.* Newline characters should not be treated the same as white space characters, because the newline character (or pair of characters) signals termination of an individual PPD entry (exceptions to this rule are noted later).

All byte codes in a PPD file must be within a range corresponding to printable 7-bit ASCII; that is, byte codes must be within the range of decimal ASCII 32 through decimal ASCII 126 inclusive, plus decimal ASCII 9 (horizontal tab), decimal ASCII 10 (line feed), and decimal ASCII 13 (carriage return). Characters that fall within this range are called *in-range byte codes*. Characters that fall outside of this range are called *out-of-range byte codes*. See section 3.7 for details on how to handle byte codes outside this range.

A *hexadecimal substring* consists of a sequence of zero or more pairs of hexadecimal digits, preceded by the < (less than) character (decimal ASCII 60) and followed by the > (greater than) character (decimal ASCII 62). Hexadecimal digits consist of the characters 0 through 9, a through f, and A through F (case is insignificant). Spaces, newlines, and tabs can be intermixed with the hexadecimal digits and should be ignored, with exceptions to this rule noted later. See section 3.7 for details on the treatment of newlines. All other characters should be considered an error. An odd number of hexadecimal digits is also an error.

## 3.3 General Parsing Summary

The following are parsing rules that apply to the PPD file as a whole:

- Any line that exceeds 255 characters in length is considered an error.
- Any character that falls outside the byte code range of decimal ASCII 32 through decimal ASCII 126 inclusive, plus decimal ASCII 9 (horizontal tab), decimal ASCII 10 (line feed), and decimal ASCII 13 (carriage return), is considered an error.

## 3.4 Main Keywords

All main keywords start with the leading special character * (decimal ASCII 42). This makes recognition of keywords easier, and reduces the possibility of keywords being confused with PostScript language identifiers in code sequences.

Query keywords start with the leading characters *?, differentiated from other main keywords by the presence of the ? character (decimal ASCII 63).

Default keywords start with the prefix *Default, as in *DefaultPageSize. Where applicable, there is a relationship between the three kinds of main keywords, as in *PageSize, *DefaultPageSize, and *?PageSize. A complete list of keywords appears later in this specification.

There is also a relationship between keywords that start with the prefix *Param, as in *ParamCustomPageSize, and the associated root keyword (*CustomPageSize, in this case). The prefix *Param signifies that this keyword documents parameters needed by the root keyword. See *CustomPageSize and *ParamCustomPageSize for more explanation.

No single keyword is wholly contained as a substring in another keyword, so that line-oriented searching programs such as *grep* can be used to parse for complete keywords, including the * as part of the keyword name. For exam-

ple, there will not be similar keywords such as *Paper and *PaperSize. However, *PageSize and *CustomPageSize are legal, because *PageSize is not a substring of *CustomPageSize.

Since the format is line-oriented, all entries will start at the beginning of a line. The * (asterisk) character that begins the main keywords in the entry must be in the first column.

Main keywords can contain any *printable* ASCII characters within the range of decimal 33 to decimal 126 inclusive, except for the characters colon, slash, space, or tab, because these characters serve as keyword delimiters.There is no escape mechanism for this prohibition, such as using double quotes to surround illegal characters (for example, *“Quoted Keyword” is not legal, because of the space in the keyword name).

The basic format of an entry looks like this

```
*Default<main keyword>:  <option_n>
*<main keyword>  <option_1>:  "PostScript language code"
*<main keyword>  <option_n>:  "some other PostScript language code"
*?<query keyword>:  "PostScript language query code"
```

An example entry

```
*DefaultPageSize: Letter
*PageSize Letter: "lettertray"
*PageSize Legal: "legaltray"
*?PageSize: "save [(Letter)(Legal)] papertray get = flush restore"
```

The information is represented as tuples. They will typically either be 2-tuples (keyword/value pairs) or 3-tuples (keyword/option/value triplets). Where simple information is supplied, such as  the name of the device, a simple keyword/value pair is used. Where there are optional parameters, 3-tuples are used (as in the example above) to provide information about a specific option.

The format conveys the possibilities for a feature: the default setting for this feature, the current setting, and how to invoke each of the options. Any lines that start with the same keyword will be contiguous in the PPD file, to make it easier to parse them. However, there is no mandatory order to the lines in an entry; for example, the query could appear above the default.

### Parsing Summary for Main Keywords

When parsing main keywords, remember

- The absence of a main keyword means that the feature does not exist (or does not make sense) on that particular device.

- The following keywords are required to be present:

  | | | |
  |---|---|---|
  | *NickName | *ModelName | *PCFileName |
  | *Product | *PSVersion | *FileVersion |
  | *FormatVersion | *LanguageEncoding | *LanguageVersion |
  | *PageSize | *PageRegion | *ImageableArea |
  | *PaperDimension | *PPD-Adobe | |

  For parsing, a chain of local customization files and included PPD files are considered one file, so the required keywords can appear anywhere in the chain of files and do not have to be repeated in each file in the chain. The absence of any of these keywords might be considered an error, or the parser might have backup strategies for handling their absence.

- If a main keyword is not recognized, the entire entry (including multi-line code segments) should be skipped. However, keep in mind that the point of the *OpenUI/*CloseUI structures is to allow new main keywords to appear without a print manager explicitly recognizing them. The most functionality will be provided to the user if a print manager handles all main keywords that occur within the *OpenUI/*CloseUI structure, displaying them and invoking their associated code to the best of its ability. Unrecognized main keywords that occur outside of the *OpenUI/*CloseUI structure should be skipped.

- A * in the first column denotes the beginning of a main keyword. Any text or white space before the * should be considered an error.

- The case of main keywords is significant. For example, *PageSize is distinct from *Pagesize. The proliferation of keywords, which are the same textually except for case is strongly discouraged.

- 40 characters is the maximum length for main keywords.

- Main keywords can contain any printable ASCII characters within the range of decimal 33 to decimal 126 inclusive, excluding the characters space, tab, colon or newline, because these characters serve as keyword delimiters.

- Delimiters for main keywords are space, tab, colon, or newline. After the initial * symbol is recognized, all characters through (but not including) the next space, tab, colon, or newline character are considered part of the main keyword.

- If a main keyword is not terminated with a colon or newline, an option keyword can be expected. See section 3.5 for information on option keywords.

## 3.5 Option Keywords

Option keywords are provided whenever there are several choices for a particular feature. For example, there might be many different media sizes listed in the *PageSize section. These choices are specified using option keywords. The option keyword immediately follows the main keyword, separated from it by one or more spaces. For example, in the following entry, the string Letter is the option keyword:

```
*PaperDimension Letter: 612 792
```

The list of option keywords is completely extensible by the person building the PPD file (generally, someone at Adobe Systems). This enables a PPD file to be generated for a device using names specified by the device manufacturer, without making constant updates to the PPD specification.

The option keywords currently known for each main keyword are registered by Adobe Systems, and their semantics are described in this specification. As new option keywords are added, updates will be generated. It should be clear, however, that the list of option keywords is never complete. That is, a new option keyword can be registered at any time. Registering the option keywords is done to prevent redundancy in naming; it is not meant to restrict the list of option keywords available.

Option keywords are composed of printable ASCII text. The following characters are forbidden, either because they are not representable within the valid byte code range or because they have special meaning:

- out-of-range byte codes
- the characters slash, colon, space, tab, or newline

An option keyword can be the name of a main keyword or of a symbol. The following examples all contain valid option keywords:

```
*InputSlot Letter: "code"
*OpenUI *InputSlot: PickOne
*SymbolValue ^MySymbol: "code"
```

An option keyword is terminated by a colon or a slash if there is a translation string (see section 3.7 for information on translation strings). There is no escape mechanism for the forbidden characters listed above.

Option keywords can have extensions called qualifiers. Qualifiers are appended to option keywords with the . (period) character (decimal ASCII 46) as a separator. Any number of these qualifiers can be appended to an option keyword, as appropriate.

The following is an example:

```
*PageSize Letter: "0 setpapertray"
*PageSize Letter.Transverse: "1 setpapertray"
*PageSize Letter.2: "2 setpapertray"
```

In this example, qualifiers are used to differentiate between several instances of a particular media type that differ only slightly. For example, the .Transverse qualifier signifies that Letter differs from Letter.Transverse only in the direction that the media is fed into the device.

The numeric qualifier .2 in Letter.2 is called a serialization extension. A serialization extension is an integer appended to an option keyword to distinguish it from an otherwise identical option keyword (for example, a device with two letter trays might refer to them as Letter.1 and Letter.2). Qualifiers will be registered when appropriate, with the exception of serialization extensions, which make no sense to register.

### Parsing Summary for Option Keywords

For print managers, the rapid extensibility of option keywords implies that a print manager should not parse for specific option keywords for two reasons:

- There might be option keywords in the PPD file that are not in this specification. New option keywords can be added to PPD files at build time when necessary. If a parser only recognizes the option keywords registered in this specification, it might limit the feature set that can be offered to the user.

- Certain option keywords might not be present in the PPD file for a given device. Manufacturers will inevitably call features by different names and use different option keywords to describe those features, so parsing for *PageSize Ledger is futile if the PPD file being parsed describes that particular feature as *PageSize 11x17. Again, this can limit the feature set offered to the user, and might cause an error if the parser cannot find a specific option keyword.

Rather than parsing for specific option keywords, a print manager should parse for main keywords and display all available option keywords found. To facilitate easier parsing, all option keywords of a given main keyword (that is conceivably part of a user interface) are bracketed by the *OpenUI/*CloseUI keywords (more on this later).

Other things to remember about parsing option keywords:

- An option keyword begins with the first character after white space after a main keyword. In other words, if a main keyword is not terminated by a colon, but is followed by white space instead, an option keyword will be the next non-white-space text encountered.

- The case of option keywords is significant. For example, letter is distinct from Letter.

- 40 characters is the maximum length for option keywords.

- Option keywords can contain any printable ASCII characters within the range of decimal 33 to decimal 126 inclusive, except for the characters colon and slash, which serve as keyword delimiters. Once the option keyword is encountered, and before it is properly terminated, a space, tab, or newline character should be regarded as an error.

- The option keyword is terminated by either a colon or a slash, which indicates the presence of a translation string. If a translation string is present, it is terminated by a colon (spaces and slashes are allowed in the translation string). A newline encountered before the colon should considered an error.

- There can be spaces and/or tabs after the colon, as in most of the examples in this specification. Spaces or tabs before the colon are interpreted as keyword delimiters, if appearing before the option keyword; as errors, if appearing within the option keyword; or, if appearing after a slash, as part of a translation string.

### 3.6 Syntax of Values

The : (colon) character(decimal ASCII 58) is used to separate keywords (and options, if any) from values. Any number of tabs and spaces are permitted after the colon and before the value.

Values can contain any printable ASCII characters within the range of decimal 32 to decimal 126 inclusive, except for the character slash, which serves as a delimiter. The characters caret and double quote also have special meaning for some values; exceptions to their use will be noted later.

A simple key/value pair looks like this

```
*MainKeyword:  value
```

and a 3-tuple typically looks like this:

```
*MainKeyword option:  value
```

There are five basic types of values:

- InvocationValue
- QuotedValue
- SymbolValue
- StringValue
- NoValue

### InvocationValue

An InvocationValue is a value that meets the following conditions:

- Occurs only in entries where there is an option keyword present.
- Starts and ends with the double quote character " (decimal ASCII 67).
- Everything between the double quotes is treated as literal; that is, *newlines* and hexadecimal substrings are allowed and are placed in the output file to be passed on to the interpreter. In this case, a newline does not terminate the PPD file entry.
- The following characters are forbidden between the starting and ending double quote characters:

  - out-of-range byte codes
  - double quote character " (decimal ASCII 67)

  There is no escape mechanism or alternate way to represent forbidden characters.

The syntax of an InvocationValue is usable directly by the PostScript interpreter. This allows InvocationValue to be stripped out of the PPD file and placed directly into output files being generated.

### QuotedValue

A QuotedValue is a value that meets the following conditions:

- Occurs only in entries where there is no option keyword present. The exception to this rule is that the `*JCL` keywords may have an option present.
- Starts and ends with a double quote character " (decimal ASCII 67).

- Between the double quote characters, a QuotedValue consists of a sequence of literal and/or hexadecimal substrings. A hexadecimal substring is defined in section 3.2. A literal substring is a sequence of in-range byte codes, with the following characters forbidden:

  - out-of-range byte codes

  - double quote character " (decimal ASCII 67)

  - open angle bracket < (decimal ASCII 60) because this character marks the beginning of a hexadecimal substring

  - closing angle bracket > (decimal ASCII 62) because this character marks the end of a hexadecimal substring

  Unlike an InvocationValue, forbidden literal substring characters in a QuotedValue can be represented as hexadecimal substrings, bounded by opening and closing angle brackets < (decimal ASCII 60) and > (decimal ASCII 62) as defined in section 3.2. A print manager parsing a QuotedValue is responsible for converting a hexadecimal substring into a sequence of bytes, which might include out-of-range byte codes, and placing the byte codes in the output file.

- The < and > characters must be represented as hexadecimal substrings if they occur in the value as anything other than hexadecimal substring delimiters.

- The value can be intermixed literal and hexadecimal substrings. For example, the following entries both have valid QuotedValues:

  ```
  *MainKeyword: "Hi there <ABCDEF> everybody"

  *MainKeyword: "<ABCDEF>"
  ```

*Note* *As a guideline, PostScript language code should not appear in a QuotedValue, but rather in an InvocationValue. If older parsers expecting literal substrings encounter a hexadecimal substring, which is new as of the 4.0 specification, errors will probably result.*

### SymbolValue

A SymbolValue is a value that meets the following conditions:

- Starts with a caret ^ (decimal ASCII 94)

- Contains only in-range byte codes and is terminated by a newline. No white space is allowed.

- The actual text of the SymbolValue is further constrained by the requirements documented in section 5.6.

A SymbolValue is used as pointer to a body of PostScript language code (an InvocationValue). A SymbolValue can occur in an entry whether or not there is an option keyword present.

### StringValue

A value of the form StringValue meets the following conditions:

- The value is not surrounded by the double quote character.
- The first character of the value cannot be a double quote character, to avoid a parser confusing a StringValue with a QuotedValue or an InvocationValue.
- The first character of the value cannot be a caret ^ (decimal ASCII 94), to avoid confusing a StringValue with a SymbolValue.
- The value is composed of in-range byte codes (printable ASCII strings), possibly separated by white space into multiple components. It is terminated by a newline, or a slash, in the case of a translation string.
- There is no escape mechanism for forbidden characters. The following characters are not allowed in the value:
  - out-of-range byte codes
  - the character slash / (decimal ASCII 47)

A StringValue can occur in an entry whether or not there is an option keyword present.

### NoValue

A value of type NoValue meets the following conditions:

- There is no option keyword present.
- There is no value present.
- The main keyword stands alone.

### Parsing Summary for Values

When parsing values, be aware of the following:

- If there is an option keyword in an entry, and the first nonwhite-space character after the colon is a double quote, " (decimal ASCII 67), the value is an InvocationValue. The exception to this rule is that if the main keyword starts with the string *JCL, the value should be treated like a QuotedValue. See section 5.23 for a description of the *JCL keywords.

- If there is an option keyword in an entry, and the first nonwhite-space character after the colon is a caret, ^ (decimal ASCII 94), the value is a SymbolValue.

- If there is an option keyword in an entry, and the first nonwhite-space character after the colon is neither a double quote, " (decimal ASCII 67) nor a caret, ^ (decimal ASCII 94), the value is a StringValue.

- If there is no option keyword, and the first nonwhite-space character after the *colon* is a double quote, " (decimal ASCII 67), the value is a QuotedValue.

- If there is no option keyword, and the first nonwhite-space character after the colon is a caret, ^ (decimal ASCII 94), the value is a SymbolValue.

- If there is no option keyword, and the first nonwhite-space character after the colon is neither a double quote, " (decimal ASCII 67) nor a caret, ^ (decimal ASCII 94), the value is a StringValue.

- The value of a *Default- main keyword entry must be a StringValue and must be a string matching a valid option keyword in that entry.

- StringValues can contain spaces, because there might be multiple components of a value.

- An InvocationValue or a QuotedValue is terminated by the closing double quote, and can be followed by a translation string, indicated by a slash after the closing double quote and before the newline. If the value has a translation string, the translation string is terminated by a newline.

- A SymbolValue, StringValue, or NoValue is terminated by a newline.

- When parsing an InvocationValue or a QuotedValue, parsing should continue until the matching closing double quote is found, even if it crosses a line boundary. Line boundaries are considered significant white space within an InvocationValue or QuotedValue. That is, lines will not be broken in the middle of PostScript language tokens. An InvocationValue or QuotedValue are considered a single “token” when parsing PPD files.

- If an InvocationValue or QuotedValue breaks across a line, the *End keyword should occur as the next entry in the PPD file after the closing double quote delimiter. If it is not found, this is considered a *parse error* with a missing close delimiter. The *End keyword appears only where an InvocationValue or QuotedValue extends across a line boundary. Care should be taken to preserve the line breaks in InvocationValues and QuotedValues. This will ensure that comments within code segments will end where they were intended to end.

- All characters inside an InvocationValue are treated as literals and are placed directly in the output file. Particularly: a slash appearing within the double quotes is not treated as a marker for the beginning of the translation string, newlines do not terminate the entry, and hexadecimal substrings do not have to be specially interpreted by the parser.

- When parsing a QuotedValue, an open angle bracket signifies the beginning of a hexadecimal substring, which is terminated by a closing angle bracket. Everything between the angle brackets should be converted to byte codes, which can be out-of range, and placed in the output file. Any non-hex data found between the angle brackets is considered an error, as is an odd number of hex digits. Ignore any white space or newlines found between the angle brackets.

- A file referenced by the *Include keyword should be treated as though it were in-line in the including (local customization) file. Be prepared for nested includes. See section 2.6 for discussion on the semantics of repeated entries and keywords.

## 3.7 Translation String Syntax

There are many entries in a PPD file that can be encountered at the user level, including main keywords and option keywords displayed as selectable choices in a user interface, and messages from the device. Sometimes these keywords and device messages can be cryptically worded and must be reworded for clarity, or they might need to be translated into another language for the user to understand them.

If keywords and messages changed with each translation of the PPD file to a new language, a parsing program would have to be written to recognize the keywords in each new language, which would greatly expand the size of the parser and the amount of work involved in writing it. Instead, a syntax is provided for the optional use of *translation strings*, which are appended to the original keywords and messages. Thus, normal keyword searches can be carried out, and the translation strings can optionally be presented to users instead of (or in addition to) the keywords.

If the PPD file is translated into several languages, there will be one PPD file for each language. There will be a main keyword entry in the PPD file that identifies the language of the translation strings to make clear in which language and encoding these strings are expressed. In various language versions of a PPD file, only the translation strings, and possibly the comments, will differ. All other information, including main keywords and option keywords, will remain the same.

A translation string can occur after an option keyword or after any type of value except a SymbolValue. A translation string is detected by the presence of a slash (*/*) character (decimal ASCII 47), and continues until a colon (if the translation string is on an option keyword) or a newline (if the translation string is on a value) is encountered.

The following is an example of the translation string syntax showing both the translation from English into French of an option keyword (Ledger) and a value (the message “out of paper”):

```
*LanguageVersion: French
*PageSize Ledger/Papier Ledger: "statusdict /ledgertray get exec"
*PrinterError: "out of paper"/Il n’y en a plus de papier.
```

This example shows the translation of an option keyword and a cryptic message into strings that are more meaningful to the user (for example, a “translation” into English):

```
*LanguageVersion: English
*PageSize Letter/Portrait Letter: "letter"
*PrinterError: "CVR OPN"/cover open
```

The presence of translation strings in a PPD file is optional. If translation strings are present, the translation strings should be used for display to the user, rather than the option keywords or messages themselves. If there are no translation strings, the option keywords and values must be displayed directly as appropriate. A parser must be especially careful not to confuse a translation string following an option keyword with the PostScript language sequence that follows in the value field, after the *colon.*

*Note* *A *Default– entry would not normally have a translation string, because the *Default– entry is typically thought of as a pointer to one of the entries in a list of options, in which case the option keyword in the list provides the translation string. However, if there is a *Default– entry with no associated option list, the *Default– entry can have a translation string. See section 2.6 and section 3.4 for more information about *Default– entries.*

To unambiguously relate natural-language characters to byte codes, an encoding is specified for each language that can be used in a PPD file. These encodings are documented in the description of the *LanguageVersion keyword in section 5.2. Translation strings in these encodings often include byte

codes that are outside the valid range of byte codes allowed in a PPD file, such as characters with accents, or they may include characters that conflict with the translation string syntax, such as translation string delimiters.

To resolve these issues, a translation string can be represented partially or wholly as a hexadecimal substring surrounded by single angle brackets (< and > decimal ASCII 60 and 62). The print manager, when parsing the file, converts the hexadecimal substring to a sequence of byte codes, which might be out-of-range, and insert the sequence into the output file.

For example, the following is a legal representation in a PPD file of the Swedish translation string for the printer error message “cover open,” using a hexadecimal substring to represent the single eight-bit ISOLatin1 character “Odieresissmall.”

```
*LanguageVersion: Swedish
*PrinterError: "cover open"/lucka <F6>ppen
```

Here is the same message, with the Swedish translation displayed entirely as a hexadecimal substring:

```
*PrinterError: "cover open"/<6C75636B61 20 F67070656E>
```

The following characters must be represented as hexadecimal substrings:

- All byte codes outside the valid range, as described in section 3.2.
- The character colon : (decimal ASCII 58, if the translation string is on an option keyword.
- The characters < and > (decimal ASCII 60 and 62), if they are part of the actual text of the translation string.

### Parsing Summary for Translation Strings

When parsing option keywords and values, remember:

- The translation string is optional (there might be 0 or 1 translation string). All parsers should be written to permit them without requiring them. If present, translation strings should be used for display to the user.
- If present, the translation string consists of a sequence of literal and/or hexadecimal substrings.
- A literal substring is a sequence of in-range byte codes, except that it cannot contain the following characters: newline, < (decimal ASCII 60) and > (decimal ASCII 62). Additionally, a colon is forbidden when the translation string is on an option keyword, because an option keyword is terminated by a colon.
- A hexadecimal substring is as defined in section 3.2 except that in a translation string, a newline in a hexadecimal string is illegal, since a newline terminates the translation string.
- The translation string begins with the first character immediately after the slash, even if it is white space. Note that the slash and white space characters are permitted in a literal substring.
- If the translation string occurs before a colon (that is, on an option keyword), it is terminated by a colon (:) or a newline. However, a newline encountered after an option keyword and before the colon will violate the syntax of option keywords.
- If the translation string occurs after a colon (that is, on a value), it is terminated by a newline.
- Out-of-range byte codes should be considered an error.

## 3.8 Human-Readable Comments

Comments are supported in the PPD files using the main keyword ‘*%’. Anything following this main keyword (through the end of the line on which it appears) should be ignored by a parsing program. The * character is the same introductory symbol used for all main keywords, and the % character is borrowed from PostScript language syntax as its comment character. These comments will begin only in column one, for simplicity.

There can also be comments in any PostScript language code, using the standard syntax of starting the comment with a %. Comments in code should be kept to a minimum, however, to reduce transmission time.

## 3.9 PostScript Language Sequences

The PostScript language sequences supplied for invoking device features are usually represented as InvocationValues. Sometimes they are represented as QuotedValues, for example, when they contain binary data.

For multiple-line InvocationValues or QuotedValues, the main keyword *End is used as an extra delimiter to help line-extraction programs (such as *grep* or *awk* in UNIX). The keyword *End also makes the PPD file more easily readable by humans, because the double quote delimiter is sometimes difficult to see at the end of a long string of code. The *End keyword will not be needed by most parsers, since the delimiting double quotes uniquely define the code segment.

*End is used only when the code requires more than one line in the PPD file. The following are two examples, one of which fits on one line, the other of which is an "extended" code sequence:

```
*PageSize Legal: "serverdict begin legaltray end"

*?PageRegion: "save
   newpath clippath pathbbox
   4 -1 roll =
   3 -1 roll =
   exch = = flush restore "
*End
```

The PostScript language sequences supplied in the PPD files are guaranteed to work only on the device for which the file was prepared. The sequences assume the default state of the interpreter. Only **userdict** and **systemdict** (and **globaldict** on Level 2 devices) are assumed to be on the dictionary stack. There will be no memory use (**save** and **restore** are used where appropriate) except as in setting frame buffers, where memory use is necessary.

## 3.10 PPD File Structure

To enable parsing, there is some minimal structure to a PPD file.

The first line of a PPD file must be

```
*PPD-Adobe: "nnn"
```

where the value "nnn" is a real number that designates conformance to a version of the PPD specification. (See section 5.5 for details on this keyword.) Files conforming to this version of the specification would have the following entry:

```
*PPD-Adobe: "4.1"
```

This line is generally followed by comment lines containing copyright and licensing restrictions.

Certain keywords are required in a PPD file. Required keywords are marked as such in their individual descriptions in this document and are listed in Parsing Summary for Main Keywords on page 15. By convention, the following *subset* of required keywords generally appears immediately after the copyright, in any order. This general information is often needed by print managers, and parsing the PPD file may be faster if the following information is included near the beginning of the file:

| | | |
|---|---|---|
| *PPD-Adobe | *NickName | *ModelName |
| *Product | *PSVersion | *PCFileName |
| *FormatVersion | *LanguageEncoding | *LanguageVersion |
| *FileVersion | | |

*ShortNickname is not required, but if it is present, it must occur before *NickName.

# 4 Syntax of Specification

Throughout this specification, certain syntactical conventions are followed to make things clearer for the reader.

## 4.1 General Syntax

The following notation is used to describe keywords.

- Main keywords, option keywords, and actual values always appear in sans serif type: *MainKeyword:, True, Null.

- Placeholder items (which will be replaced by an actual value in the PPD) appear in sans italic type: *mediaOption, invocation.*

- The vertical bar ( | ) character is used to mean “or”, where “or” is an exclusive or. For example, this entry in the PPD specification:

  ```
  *DefaultManualFeed: True | False
  ```

  This entry in the PPD file will read either *DefaultManualFeed: True or *DefaultManualFeed: False, but True and False cannot both appear in this entry.

- The ellipsis ( ... ) means that more than one instance of a token can appear, separated by white space. For example, this entry in the PPD specification

  ```
  *Extensions: extension...
  ```

This main keyword has several possible values, indicating which language extensions are supported by the device. Because a device can support several language extensions, this keyword can have multiple values, separated by white space.

For example, both of these PPD file entries are valid

```
*Extensions: FileSystem
*Extensions: CMYK FileSystem Composite
```

† Note that a few of the main keywords may require exiting the server loop for correct execution. These have been flagged by a dagger in the left margin as shown here. The print manager is responsible for obtaining the value supplied by the *Password entry, or an alternate password as supplied by system software or the user, and prepending it to the code sequence that is the value of the flagged entry. The code contained in these flagged entries must check for the existence of a valid password on the operand stack when executing.

UI Main keywords that will be bracketed with the *OpenUI/*CloseUI keywords in PPD files built by Adobe Systems will be marked in this specification with this symbol (for "user interface") by the keyword definition. See section 5.5 for more details on *OpenUI/*CloseUI.

## 4.2 Elementary Types

The PPD specification employs various elementary types of expressions. These types are defined in this section.

***filename***

A *filename* is a QuotedValue and is subject to the rules of QuotedValues.

Currently, filename is used only by the *Include keyword. It can be the name of the file itself, or it might be a path to the file. The following are all examples of legal filenames:

```
*Include: "MyDevice.PPD"
*Include: "/home/adobe/PPDfiles/myfile.ppd"
*Include: "My<3C>test<3C>file.ppd"
*Include: "C:\lib\MyDevice.PPD"
```

In the third example, the filename contains the double quote character. This character is represented as a hex string, according to the rules of a QuotedValue. The encoding of a filename is system-dependent and is not necessarily portable to other systems. At minimum, the filename or pathname might have to be edited when porting.

***fontname***

A fontname is a simple text string, subject to the rules defined in section 3.3. A fontname is delimited by blanks. Examples of standard fontnames can be found in Standard Character Sets and Encoding Vectors, in Appendix E of the PostScript Language Reference Manual, Second Edition, and some are listed here:

```
Times-Roman
Helvetica-Bold
NewCenturySchoolbook-Italic
```

Notice that fontname does not start with a slash character (/) as it does in the PostScript language when the fontname is specified as a literal.

***int***

An *integer*, as used in this specification, is a non-fractional number that has no sign. There are practical limitations for an *integer*'s maximum value, but as a default it should range between 0 and $4.295 \times 10^9$ (32 bits).

***invocation***

An *invocation* is an arbitrary sequence of valid PostScript language commands. An invocation is generally used to perform some manipulation of the device. It can be represented either as a QuotedValue or as an InvocationValue, depending on the keyword described.

***JCL***

*JCL* is an arbitrary sequence of valid job control language commands. This code is generally used to perform some manipulation of the device outside of the control of the PostScript interpreter. It is represented as a QuotedValue because it may contain out-of-range byte codes.

***option***

An *option keyword*, or simply *option*, is a text string subject to the rules defined in section 3.3. In this specification, the placeholders for option keywords are generally preceded by some sort of nominal qualifier, such as *mediaOption* or *trayOption*.

***query***

A *query* is an arbitrary sequence of valid PostScript language commands that returns one or more values to the host via the reverse channel of the device. Queries are used to request information from the device about the current state. Queries only work on devices that are connected to a host by a bidirectional channel, so information can be returned to the host.

*Note* *Because of its format, the value of a query is a QuotedValue, but it is strongly recommended that writers of PPD files follow the rules for InvocationValues when writing queries. Older parsers might produce errors if faced with the hexadecimal substrings and other odd characters allowed in QuotedValues but not in InvocationValues.*

***real***

A *real* number is a fractional number that can be signed or unsigned. There are practical limitations on the maximum size of a *real*, but as a default it should range between $3.4 \times 10^{-38}$ to $3.4 \times 10^{+38}$.

***string***

A *string* is comprised of any printable characters. It is delimited by blanks except when it is specifically stated that the string may contain blanks, in which case another set of delimiters is also stated. For example, in translation strings, the delimiters are a preceding slash and a succeeding colon or newline, and the string may contain blanks.

```
Thisisatextstring
keyword/This is a translation string:
```

The length of a string is limited by the 255 characters-per-line limit described in section 3.3.

## 4.3 Sample Keyword Entries

The format of this section shows the main keyword, the possible option keywords, and a pseudo-code syntax to illustrate its value. A main keyword can have either a restricted option keyword list (option keywords are restricted to those listed), an unlimited option keyword list (option keywords can be added at any time), or no option keywords at all.

The value type of each keyword (InvocationValue, QuotedValue, SymbolValue, StringValue) is recorded in the description of the keyword, with the following exceptions:

- Query keywords, which are always of type QuotedValue
- *Default- keywords, which are always of type StringValue

Here are examples of what each type of entry looks like in this specification:

***MainKeyword** Option1 | Option2: *"invocation"*

This indicates that for the main keyword *MainKeyword, there are two viable option keywords (Option1, Option2), and the appropriate syntax for the value of the tuple is a PostScript language invocation string enclosed in double quotes. In the PPD file, there would be one entry for each main keyword-option keyword pair:

```
*MainKeyword Option1: "invocation"
*MainKeyword Option2: "invocation"
```

A typical example of a restricted option list would be a keyword whose only options are True and False:

```
*Collate True: "1 dict dup /Collate true put setpagedevice"
*Collate False: "1 dict dup /Collate false put setpagedevice"
```

***MainKeyword** *optiontype*: *"invocation"*

This indicates that for the keyword *MainKeyword, there can be many viable option keywords. The currently known option keywords will be listed in this specification with the main keyword, but others can be added at any time. As above, the appropriate syntax for the value of the tuple is a PostScript language invocation string enclosed in double quotes. Again, in the PPD file, there would be one entry for each main keyword-option keyword pair:

```
*DifferentKeyword Option1: "invocation"
...
*DifferentKeyword Optionn: "invocation"
```

For example, the list of page sizes offered by a device is an extensible list:

```
*OpenUI *PageSize: PickOne
*DefaultPageSize: Letter
*OrderDependency: 30 AnySetup *PageSize
*PageSize Letter: "lettertray"
*PageSize Legal: "legaltray"
*PageSize Ledger: "ledgertray"
*PageSize A4: "a4tray"
*PageSize B5: "b5tray"
*?PageSize: “query code”
*CloseUI: *PageSize
```

***MainKeyword:** *"int"*

This main keyword has no option keywords, and the appropriate syntax for the value of the tuple is an integer enclosed in double quotes. For example

```
*FreeVM: "110980"
```

# 5 Keywords

This section contains a list of the currently defined keywords and a description of their uses. The keywords are grouped according to their functionality. Where appropriate, registered option keywords are documented along with the keywords with which they are associated. This is to ensure that there will never be two different PPD files containing the same option keyword, where the keyword has different semantics in each file.

If there is no option keyword describing a particular aspect of a documented feature, then the device manufacturer can request a new option keyword from Adobe. If an option keyword already exists to describe a feature, it will be used.

All keywords are optional in a PPD file, unless noted as *Required* in the keyword description.

*Note* *If a feature is not supported by a device, that feature's default, invocation, and query are omitted from the PPD file. Absence of a feature in the PPD implies lack of device support for that feature.*

## 5.1 Standard Option Values for Main Keywords

The following option keywords are used with many different main keywords and have universal meaning throughout a PPD file.

### True

True is used in a PPD file in two ways. When used as the value of a *Default keyword, True means that the default state of that particular feature is on. For example, the following entry means that this device will feed media from the manual feed slot unless explicitly told to do otherwise.

```
*DefaultManualFeed: True
```

When used as an option to a main keyword, True means that the value of that option of the keyword is the PostScript language code required to "turn on" or invoke the feature. For example, the following entry contains the code to enable the manual feed slot:

```
*ManualFeed True: "statusdict /manualfeed true put"
```

### False

Like True, False is used throughout a PPD file in two ways. When used as the value of a *Default- keyword whose value is a boolean True or False, False means that the default state of that particular feature is "off." For example, the following entry means that this device will not feed media from the manual feed slot unless explicitly told to do so.

```
*DefaultManualFeed: False
```

When used as an option to a main keyword, False means that the value of that option of the keyword is the PostScript language code required to "turn off" or disable the feature. For example, the following entry contains the code to disable the manual feed slot.

```
*ManualFeed False: "statusdict /manualfeed false put"
```

### None

Like False, None is used to indicate that a certain feature is disabled (off) by default, and also to indicate how to disable (turn off) a feature. False is used with boolean features; None is used for features with more than two states. For example:

```
*DefaultFoldType: None
*FoldType None: "code to turn off folding"
*FoldType Saddle: "code to invoke a saddle fold"
*FoldType ZFold: "code to invoke a z-gate fold"
```

Code associated with a None option will explicitly disable the feature in question. In the example above, the None option would contain code to explicitly invoke "no folding."

*Note* *None is never used to indicate the absence of a feature. If a feature is absent, the feature's keywords will be omitted from the PPD file. For example, if a device does not support manual feed, the manualfeed keywords are omitted entirely. *DefaultManualFeed: None is invalid.*

### Unknown

This string is returned from queries if the correct information can not be determined, or none of the valid keywords can be returned. It is also used as an option keyword with *Default- keywords to denote that there is no specific default state (for example, *DefaultPageSize: Unknown on a device whose page size is not set until the user requests a page size.)

*Note* *Like None, Unknown is not used to indicate complete absence of a feature; if a feature is absent, the feature's keywords will be completely omitted from the PPD file.*

## 5.2 General Information Keywords

The keywords in this section provide general information about the PPD file and the device it describes. The keywords in this section do not invoke any device features.

***FileVersion:** *"string"*

*Required*. This keyword identifies the version number of the PPD file itself. It is used only to distinguish between releases of the same file, not to distinguish one file from another.

The value, a QuotedValue, is a string of the form "1.0. A standard version numbering scheme is employed. For major changes to the device and the PPD file, the entire number will be incremented to the next whole number (for example, from 1.0 to 2.0). For minor fixes to the PPD file (including typographical errors), the integer to the right of the decimal will be incremented (for example, from 1.0 to 1.1). This permits the various versions of a PPD file to be identical in most ways (including file name) but still be distinguished from one another. All released PPD files will initially have the string "1.0" in this field

***FormatVersion:** *"string"*

*Required*. This provides the version number of the PPD file format specification to which the PPD file conforms. It is retained primarily for backward compatibility, as the newer keyword *PPD-Adobe: "4.1" provides both the information that this file is a PPD file *and* the specification version to which the file conforms.

The value, a QuotedValue, is a string of the form "1.0." A standard version numbering scheme is employed, where digits to the left of the decimal imply incompatible changes, and digits to the right of the decimal imply minor revisions. Any revisions to the format will be documented in an update to this specification. For a PPD file to conform to the version of the specification detailed in this document, the value of *FormatVersion must be "4.1".

***LanguageEncoding:** *encodingOption*

*Required*.This keyword complements and partially supercedes the older *LanguageVersion keyword (see page 38 for a description of *LanguageVersion). *LanguageEncoding identifies the encoding (mapping from natural language characters to byte codes) used in the human-readable comments, translation strings, and certain QuotedValues such as the value of *NickName. The encoding of any part of the PPD file other than these strings is system-dependent.

*LanguageEncoding does not identify the natural language of the PPD file; that is the role of the *LanguageVersion keyword. In most cases, a parser needs only to parse the *LanguageEncoding keyword, and *LanguageVersion can be ignored. The value of *LanguageEncoding contains the information needed to allow a parser to convert text strings from the encoding used in the PPD file to the encoding used on the host system.

If *LanguageEncoding is present, its value overrides the default encoding implied by the *LanguageVersion keyword.

The values of *encodingOption*, a StringValue, are as follows:

- ISOLatin1—Uses the ISOLatin1 encoding
- JIS83-RKSJ—Uses the RKSJ (informally known as “Shift JIS”) encoding and the JIS X0208-1983 character set
- MacStandard—Uses Macintosh® standard encoding
- WindowsANSI—Uses Windows® ANSI encoding, as defined by Microsoft® for use in the Windows operating system
- None—The encoding is not specified

Appendix C provides tables to convert between the ISOLatin1, MacStandard, and WindowsANSI encodings.

Translation strings and QuotedValues may include byte codes that are outside the valid range allowed in a PPD file. To resolve this, a translation string or QuotedValue can be represented wholly or partially as a hexadecimal substring. See section 3.7 for details on translation string syntax.

***LanguageVersion:** *languageOption*

*Required*. This identifies the natural language used in the PPD file. For simplicity, the valid values of *languageOption* are the English words for the natural languages. The value of *languageOption* (for instance, French or German) affects only the human-readable comments, translation strings, and certain QuotedValues such as the value of *NickName. The encoding (mapping from natural language characters to byte codes) of any part of the PPD file other than these strings is system-dependent.

The *LanguageEncoding keyword (defined on page 37) specifies the encoding for the strings mentioned above. If *LanguageEncoding is absent, the encoding of these strings can be deduced from the value of *LanguageVersion. The currently registered values for *languageOption*, which is a StringValue, and their corresponding encodings (defined under *LanguageEncoding) are:

**Table 1** *Values for languageOption and their encodings*

| *languageOption* | *encoding* |
|---|---|
| **English** | ISOLatin1 |
| **Chinese** | None |
| **Danish** | ISOLatin1 |
| **Dutch** | ISOLatin1 |
| **Finnish** | ISOLatin1 |
| **French** | ISOLatin1 |
| **German** | ISOLatin1 |
| **Italian** | ISOLatin1 |
| **Japanese** | JIS83-RKSJ |
| **Norwegian** | ISOLatin1 |
| **Portuguese** | ISOLatin1 |
| **Russian** | None |
| **Spanish** | ISOLatin1 |
| **Swedish** | ISOLatin1 |

***ModelName:** *"string"*

*Required*. This value, a QuotedValue, is a string created by the builder of the PPD file that represents the common name of the device. It is unique for a given model of device, but not for a given instance of that model. Because *Product is not always unique or descriptive, *ModelName can be used to

identify a PPD file for a specific device model. See the description of *Product for an example of how *ModelName, *NickName, and *Product are used together.

Because *ModelName is used as a base for the PPD file name in some environments, certain punctuation characters are illegal. The value of *ModelName can contain only the following:

- alphanumeric characters
- space (decimal ASCII 32)
- period, . (decimal ASCII 46
- hyphen, - (decimal ASCII 45)
- slash, / (decimal ASCII 47)
- plus, + (decimal ASCII 43)

No other punctuation characters are allowed.

Because *ModelName describes a unique printer model, and because it may be used as a filename in some environments, there should be only one *ModelName entry per PPD file. If a PPD file describes two or more models, that fact should be reflected in the value of *ModelName. For example:

```
*ModelName: “Acme FunPrinter or NiftyPrinter”
```

***NickName:** *“string”*

*Required*. This value, a QuotedValue, is the local name for the device. It is unique for an instance of a device model. It is used primarily at the user interface level when selecting a device or to distinguish between two otherwise indistinguishable devices (for example, if a single controller is used to drive more than one type of marking engine).

Initially, the value of *NickName is usually the same as the value of *ModelName, but it can be edited in a local customization file, to reflect a local instance of a printer. Alternatively, the value may have a translation string for localization. See the description of *Product for an example of how *ModelName, *NickName, and *Product are used together.

Note that the value of *NickName, as a QuotedValue, can include hexadecimal substrings. These substrings should be translated to natural language characters according to the values of *LanguageEncoding and/or *LanguageVersion.

Because *NickName can be used to describe a single instance of a unique printer model, there should be only one *NickName entry per PPD file. The length of the value of *NickName is unrestricted. For situations where the length of the nickname must be restricted, see the description of *ShortNickName on page 42.

***PCFileName:** *"string"*

*Required*. This value, a QuotedValue, provides the name of the PPD file as it would appear in a PC environment. This name is limited to eight characters followed by a dot and a three character suffix. It is provided in the PPD file so that a PPD file with a longer name, transferred from another platform, can be renamed to a unique PPD filename appropriate to the PC environment.

***Product:** *"(string)"*

*Required*. This value, a QuotedValue, corresponds exactly to the product string of the device. On Level 1 devices, it is the value returned by the code sequence

```
statusdict /product get exec == flush end
```

and on Level 2 devices, the value is returned by this code sequence

```
product == flush
```

There can be more than one instance of the *Product keyword if the PPD file is valid for more than one product.

*Product, *NickName, and *ModelName are used together in the following manner. In this example, *NickName has been customized by a system administrator at a local site:

```
*Product: "(LaserPrinter)"
*ModelName: "Acme SuperPrinter"
*NickName: "Joe's Printer in Room 101"
```

***PSVersion:** *"(string) int"*

*Required*. This QuotedValue is composed of two parts. The string in parentheses is the version number of the PostScript interpreter, as returned by the code sequence

```
version == flush
```

The integer following the parentheses is the interpreter's revision number. On Level 2 devices, this number is returned by the code sequence

```
revision == flush
```

On Level 1 devices, the revision number is returned by the code sequence

```
statusdict /revision get exec == flush
```

The values are presented in PostScript language form so that they can be compared with the actual values in the device to determine whether or not the PPD file matches the device.

There can be more than one instance of the *PSVersion keyword in a PPD file if the PPD file is valid for more than one version (and revision) of the interpreter. For example, when the PPD file is built, it might be known that the PPD files for version 52.3 revision 1 and for version 54.0 revision 0 of the interpreter on that device are identical, so they can be combined into one PPD file with the following entries:

```
*PSVersion: "(52.3) 1"
*PSVersion: "(54.0) 0"
```

For a given device, the PPD file can be useful even if the interpreter version number in the PPD file does not match the version number of the interpreter in the device. For example, the device's interpreter can be upgraded with faster font rasterization algorithms. The upgraded interpreter would have a new version number, but, because faster algorithms are not reflected in the PPD file, a new PPD file is not necessary. Therefore, as long as the change to the interpreter does not affect the PPD file, a user can safely use a newer version of the interpreter with a PPD file built for an older interpreter. Conversely, if a PPD file is available for the newer version of the interpreter, a user could use that PPD file with an older version of the interpreter.

Likewise, a match between the version number in the PPD file and the version number of the interpreter is not necessarily significant. For example, a manufacturer might use a single controller to drive several different marking engines. In this case, a PPD file should be built to describe each controller-engine combination, not just the controller alone. However, the product name, version, and revision number might be the same in each PPD file.

For example, imagine a controller called SuperRIP, which can drive two different engines called the S2500 and the S5000. The SuperRIP controller contains interpreter version 50.3, and the product name is always "SuperRIP," regardless of which engine is attached to the controller. In this case, there would be two PPD files: one for the combination of SuperRIP and the S2500 engine, and one for the combination of SuperRIP and the S5000 engine. These PPD files would be called, respectively, *S2500503.PPD* and *S5000503.PPD*.

The relevant entries in *S2500503.PPD* would look like this:

```
*PSVersion: (50.3) 1
*Product: "(SuperRIP)"
*NickName: "SuperRIP with S2500 v.50.3"
```

and the relevant entries in *S5000503.PPD* would look like this

```
*PSVersion: (50.3) 1
*Product: "(SuperRIP)"
*NickName: "SuperRIP with S5000 v.50.3"
```

These two PPD files, while matching in product name and interpreter version number, are differentiated by their *NickName entries and their filenames. The real responsibility of matching PPD files with physical devices lies with the system administrator and/or system software that uses these files. The process might be to simply ask the user to select one, or the installation process might make the proper association between PPD file and device.

***ShortNickName:** *"string"*

This keyword is identical to the *NickName keyword except that the length of the string value is limited to 31 or fewer characters. This keyword is provided to overcome certain string length restrictions in some host environments.

The value, a QuotedValue describing the device, must be unique within the set of PPD files on the local system. That is, there should not be two different PPD files with the same value for *ShortNickName. Also, there should be only one instance of *ShortNickName per PPD file.

Generally, *ShortNickName will only appear in a PPD file when the value of *NickName is longer than 31 characters. If *ShortNickName is present, it must occur in the PPD file before the instance of *NickName.

## 5.3 Basic Device Capabilities

The keywords in this section provide information about the device's basic capabilities.

***ColorDevice:** True | False

This keyword indicates whether or not the device physically outputs color. See *Extensions for information about black and white devices that support the color extensions to the PostScript language. The value is of type StringValue.

***DefaultColorSpace:** *colorspaceOption*

This keyword indicates the default native color space of the device. The native color space is the color space that all colors are converted into before rendering. The currently registered values for *colorspaceOption* (a StringValue) are

- CMY—This device uses the cyan-magenta-yellow color space as its native color space.
- CMYK—This device uses the cyan-magenta-yellow-black color space as its native color space.
- RGB—This device uses the red-green-blue color space as its native color space.
- Gray—This device uses a gray-scale native color space.

***Extensions:** *extensionOption ...*

This keyword indicates that this device supports the PostScript language extensions listed. One or more extensions may be listed, separated by white space. Operators specific to each extension are documented in Appendix A of the *PostScript Language Reference Manual, Second Edition*.

The currently registered values for *extensionOption* (a StringValue) are

- DPS—This device contains a PostScript Level 1 implementation that also supports the Display PostScript™ Extensions.
- CMYK—This device contains a PostScript Level 1 implementation that also supports the Color Extensions
- Composite—This device contains a PostScript Level 1 implementation that also supports the Composite Font Extensions
- FileSystem—This device contains a PostScript Level 1 implementation that also supports the File System Extensions

***FaxSupport:** *faxOption*

If the device can act as a facsimile (fax) device, this keyword lists the various fax-related capabilities of the device. One or more capabilities can be listed, separated by white space.

Currently, the only registered value (a StringValue) is

- Base—This device can encode the rasterized version of a document in fax format and transmit the fax to another fax device.

***FileSystem:** True | False

This StringValue indicates whether or not the PostScript device has the capability for a writable file system. Normally this means the presence of a hard disk or SCSI controller on the device. This information can be used by a printing manager to determine the capability for internal file system support. Note that some devices might have the capability for a file system but might not in fact have a disk installed (in this case the value for this keyword would be True, but the associated query would return False). The *?FileSystem query can be used to dynamically determine whether or not a file system is actually present. If the device has no capability of having a file system, this entry can be omitted.

***?FileSystem:** *"query"*

This query will return True if a writable file system is currently online, False if not, and Unknown if the state cannot be determined. The results of this query do not convey any information about whether or not the disk is initialized, or how many free pages there are. If this device cannot support a file system, this entry will be omitted.

***LanguageLevel:** *"int"*

This QuotedValue designates the PostScript language level supported by the PostScript interpreter in this device. If the value is 2, the PostScript interpreter in this device supports all PostScript Level 2 features. If the value is 1 or if this keyword is not present, the PostScript interpreter supports all PostScript Level 1 features. See *Extensions for further information.

***Throughput:** *"int"*

This QuotedValue is the nominal throughput in pages per minute. It represents the marking engine's rated capacity for throughput. It might be used to determine the fastest of a number of devices if there are many to choose from, but should not be construed as any kind of "benchmark" figure.

In the case of roll-fed machines, the number indicates the number of 8-1/2 inch sections of media that can be fed in one minute by the marking engine. In the case of duplex devices, which can print on both sides of the paper, the

number indicates the number of pages that can be printed in one minute in simplex (one-sided) mode. If the value is fractional, it is rounded up to the nearest number (it must be 1 or larger).

***TTRasterizer:** *rasterizerOption*

This keyword indicates whether or not this device contains software to create font bitmaps from Type 42 (TrueType™) font outlines. If the device does contain such "rasterizer" software, the *rasterizerOption* indicates whether the software is built into the device, is downloadable, or other details.

The currently registered values (of type StringValue) for *rasterizerOption* are

- None—This device does not contain a Type 42 rasterizer and the device is not capable of receiving a downloaded rasterizer.
- Accept68K—This device does not contain a Type 42 rasterizer, but the device can accept a downloaded rasterizer that is 68000-compatible. A driver wishing to download a rasterizer should also query the current state of free VM on the device to determine whether there is enough memory to accept the rasterizer.
- Type42—This device contains a Type 42 rasterizer in ROM.

***?TTRasterizer:** *"query"*

This query returns the *rasterizerOption* corresponding to the device's capability regarding Type 42 rasterizer software. The value returned must be one of the *rasterizerOptions* listed under *TTRasterizer or it will be Unknown. If Accept68K is returned by this query, a parser should also query the current state of free VM to determine whether there is enough memory to download the rasterizer.

## 5.4 Emulations and Protocols

The keywords in this section provide information about emulators and protocols supported by the device.

***Protocols:** *protocolOption ...*

This provides a StringValue that indicates the protocols supported by this device. One or more extensions can be listed, separated by white space.

Valid values for *protocolOption* are:

- BCP—This device supports the Adobe binary communications protocols, as documented in section 3 of the Technical Specification, "Adobe Serial and Parallel Communications Protocols Specification", available from the Adobe Developers Association. The binary communications protocol provides a clear channel on a serial or parallel line and is used to transparently pass certain control characters that might be contained in binary data. On a clear channel, switching between the PostScript language and certain emulators can be accomplished transparently using language commands from within a job.

- PJL—This device can support multiple printer languages, including the PostScript language. Hewlett Packard's printer job language (PJL) provides a means of switching between languages. This device supports the PJL language switching sequences that begin and end PostScript language jobs.

*Note: PPD files that conform to specification version 4.1 and higher and that contain this entry will also contain the *JCL keywords that provide the appropriate PJL language switching sequences. In version 4.0 PPD files, the *JCL keywords do not exist and the print manager must provide the appropriate PJL commands.*

- TBCP—This device supports the tagged binary communications protocol, as documented in section 4 of the Technical Specification, "Adobe Serial and Parallel Communications Protocols Specification", available from the Adobe Developers Association.

***Emulators:** *emulatorOption ...*

This keyword provides a StringValue that enumerates the emulators that can be invoked from within a PostScript language job on this device. The value consists of one or more *emulatorOption* keywords.

For each *emulatorOption* present, corresponding main keywords may also be present which describe the command sequence necessary to start and stop the emulator named *emulatorOption*. These main keywords are formed by concatenating the strings *StartEmulator_ and *StopEmulator_ with the *emulatorOption* string.

For example:

```
*Emulators: hplj proprinter
*StartEmulator_hplj: "code"
*StopEmulator_hplj: "code"
*StartEmulator_proprinter: "code"
*StopEmulator_proprinter: "code"
```

An *emulatorOption* must appear in a *Emulators entry before the corresponding *StartEmulator_ and *StopEmulator_ keywords appear. Multiple *Emulators entries may appear, and their contents should be accumulated. For each *emulatorOption* listed under *Emulators, there must be corresponding *StartEmulator_ and *StopEmulator_ entries.

The currently registered option keywords for *emulatorOptions* are

- diablo630—Diablo 630
- decppl3—Digital ANSI-Compliant Printing Protocol (level 3)
- hpgl—Hewlett Packard Graphics Language
- hplj—Hewlett Packard LaserJet and LaserJet Plus (HP-PCL)
- proprinter—IBM ProPrinter
- ti855—Texas Instruments 855

***StartEmulator_emulatorOption:** *"invocation"*

This QuotedValue provides the PostScript language code to invoke the emulator named *emulatorOption*, from within the current job. The invocation is represented as a QuotedValue in case the invocation code contains 8-bit control characters, which must be represented as hexadecimal substrings. This keyword is formed by concatenating the string *StartEmulator_ with the string from the list of valid *emulatorOptions* listed under *Emulators.

The code in the QuotedValue must end with a space or newline, so that the final PostScript language token is executed. Any data sent by the print manager following the invocation code will be executed by the named emulator. For example:

```
*Emulators: hplj
*StartEmulator_hplj: "currentfile /hpcl statusdict /emulate get exec "
```

Before invoking any emulators, a clear channel must be established. See the description of the keyword *Protocols for more information about establishing a clear channel.

*Note* *Before beginning an emulation, most emulators will erase the current page, initialize the graphics state, and clear the operand and execution stacks.*

***StopEmulator_emulatorOption:** *"hexadecimal data"*

This QuotedValue provides the data needed to exit the emulator named *emulatorOption* and return to PostScript interpretation. The invocation is represented as a QuotedValue because typically the code contains control characters, which must be represented as hexadecimal substrings.These hexadecimal substrings should be parsed by the print manager, and the appropriate 8-bit characters should be sent to the device.

This keyword is formed by concatenating the string *StopEmulator_ with the a string from the list of valid *emulatorOptions* listed under *Emulators. For example:

```
*Emulators: hplj
*StartEmulator_hplj: "currentfile /hpcl statusdict /emulate get exec "
*StopEmulator_hplj: "<1B7F>0"
```

## 5.5 Structure Keywords

The keywords in this section provide structure to the PPD file by enclosing certain groups of keywords to provide added meaning to a parser. The keywords in this section do not invoke any device features. Instead, they are intended to assist with parsing the PPD file and building a user interface.

***PPD-Adobe:** "4.1"

*Required*. This keyword must be the first line in a valid PPD file. The 4.1 at the end signifies conformance to the 4.1 version of this specification. Conformance to a later version of this specification would be indicated by a higher number. This entry is of type QuotedValue.

***OpenUI** *mainKeyword*: PickOne | PickMany | Boolean

***CloseUI:** *mainKeyword*

When parsing a list of options, it is convenient for a parser to distinguish between main keywords that refer to features commonly displayed in a user interface for selection by a user, and main keywords that provide information about the device. To this end, *OpenUI and *CloseUI are used to bracket all the information about a given main keyword, if that main keyword describes device features that can be selected by the user.

For example, the list of page size options offered by the *PageSize keyword will be bracketed with *OpenUI/*CloseUI, because the device supports the user selection of a page size, but the keyword *Throughput, which describes the rated printing capacity of the device, would not be bracketed by *OpenUI/*CloseUI, because the throughput is not selectable by the user. Query keywords, *Default- keywords, and any other associated keywords, such as *Param- keywords and *OrderDependency, are included in the *OpenUI/*CloseUI bracketing for convenience and readability.

The StringValue values PickOne, PickMany, and Boolean denote the type of option list for this feature. They assist in building a user interface that makes sense for this feature.

- PickOne means that, for this feature, the device supports only one choice at a time from the list of available options. The print manager can use this information when building a user interface, so that when a user selects any single choice from the list, the other choices are disabled. An example of a PickOne type of list is *PageSize, which displays the list of available page sizes. A page must always have a size, and it cannot be two sizes at once.

In a PickOne type of option list, the option None has special meaning. If one of the options is None, the code associated with None will explicitly disable the other options. For example, a list of stapling options might include None, meaning “do not staple.” If the user selects None, the print manager is responsible for sending the code supplied to disable stapling, in case the device is set up to staple by default.

- PickMany means that, for this feature, the device supports one or more choices at a time from the list of available options. For example, a device might allow two kinds of folding to occur at once. The print manager can use this information when building a user interface, so that when a user selects any single choice, the other choices in the list are not disabled or deselected.

  In a PickMany type of option list, one of the options must be None. The None option is an exception to the general PickMany rule of allowing more than one choice at a time. As with a PickOne list, the code associated with the option None must explicitly disable all other choices. In a user interface, if a user selects None, all other options should be deselected by the print manager. If a user selects any option other than None, then None should be deselected.

- Boolean means that there are only two choices, True and False, and thus informs a print manager that this item could be displayed as a check box or as radio buttons, rather than as a menu list.

The option of *OpenUI and the StringValue value of *CloseUI, *mainKeyword*, is the keyword whose options are listed within the *OpenUI/*CloseUI pair. It must be one of the main keywords registered in this specification or in an addendum to this specification. The value may optionally have a translation string to record the manufacturer’s preferred name for this particular feature.

The following is a sample entry:

```
*OpenUI *Smoothing/Smooth Characters: Boolean
*DefaultSmoothing: False
*OrderDependency: 40 AnySetup *Smoothing
*Smoothing True: "invocation code"
*Smoothing False: "invocation code"
*?Smoothing: “query code”
*CloseUI: *Smoothing
```

Given this entry in a PPD file, the print manager could use this information to display a menu or radio buttons labeled “Smooth Characters” with two options, True and False.

*OpenUI/*CloseUI allows for the automatic building of a user interface. The user-selectable options bracketed by *OpenUI/*CloseUI all follow a certain form. Each entry consists of a main keyword, an option keyword, and a value,

which is a code sequence. The print manager can display all options listed bracketed by *OpenUI/*CloseUI, record the user's selection of an option, extract the code related to that option, and send the code to the device.

The print manager never has to know exactly which code is being sent. It follows, then, that options could be added to a main keyword without the print manager having to be rewritten to parse for them. As long as the print manager displays and executes all options bracketed by *OpenUI/*CloseUI, options can be added to this specification without rewriting the print manager code.

Likewise, new main keywords can be parsed, if they are of the proper form that can be enclosed in *OpenUI/*CloseUI. A print manager that parses *OpenUI/*CloseUI properly should not parse for a specific main keyword, and, therefore, should not care if a new main keyword is added. Not all new main keywords fall into the *OpenUI/*CloseUI category, but if they do, a print manager should read them, display their options, and execute the associated code without having to be rewritten.

Before invoking certain features, a print manager must do extra work, such as opening a communication channel or requesting information from a user. These more complex features are not amenable to the simplistic syntax of *OpenUI/*CloseUI, so their keyword/option lists will not be bracketed with *OpenUI/*CloseUI.

UI Main keywords that will be bracketed with *OpenUI/*CloseUI in PPD files built by Adobe Systems will be marked in this specification with the UI symbol (for user interface) by the keyword definition.

*OpenUI/*CloseUI is provided as a supplementary service designed to assist in building a user interface. It is not intended to constrain a print manager or other application in any way. If a print manager does not want to display to the user a feature marked with *OpenUI/*CloseUI, the print manager can parse for that feature's keyword and not display the feature. Likewise, if a print manager wants to display additional features not marked by *OpenUI/*CloseUI, the print manager can parse for that feature's keyword and display the feature to the user.

The *OpenUI/*CloseUI provides most of what typically constitutes a user interface, so that a "dumb" parser could construct a reasonable user interface simply by reading all the features marked by *OpenUI/*CloseUI and displaying them. It is not meant to be a complete user interface design tool.

***OpenGroup:** *string*

***CloseGroup:** *string*

Like *OpenUI/*CloseUI, this pair of keywords is provided to assist in building a user interface. Because of the limited physical space of some displays, print manager writers often need a way to group certain features behind one "button" or menu item. Within a PPD file, certain classes of features lend themselves particularly well to grouping. A selection of features can be grouped by placing a *OpenGroup/*CloseGroup pair around the feature set.

Features within a group are not mutually exclusive; several features can be chosen from a group. For example, a group of finishing features might contain stapling, folding, binding, and other finishing features that are not exclusive of one another.

The StringValue value of both *OpenGroup and *CloseGroup is a descriptive string that represents the name of the group. It is provided for the print manager to display to the user.

The example below groups all of the finishing features of the device (the invocation code is completely imaginary):

```
*OpenGroup: Finishing
*OpenUI *FoldType: PickOne
*DefaultFoldType: None
*FoldType Saddle: "1 dict dup /Fold 1 put setpagedevice"
*FoldType None: "1 dict dup /Fold 0 put setpagedevice"
*CloseUI: *FoldType
*OpenUI *Collate: Boolean
*DefaultCollate: True
*Collate True: "1 dict dup /Collate true put setpagedevice"
*Collate False: "1 dict dup /Collate false put setpagedevice"
*CloseUI: *Collate
*CloseGroup: Finishing
```

The sample PPD file at the end this specification groups imagesetter features into one group. Another logical group is finishing features. By parsing for *OpenGroup/*CloseGroup, a print manager can display Finishing and Imagesetting to the user, and all the finishing features and imagesetting features can be in separate dialog boxes, hidden from the user until they are needed.

This syntax, like the syntax of *OpenUI/*CloseUI, allows features to be added to a group without the print manager needing to know about them. New *OpenUI/*CloseUI features can be added within an *OpenGroup/*CloseGroup grouping, and a print manager that parses for *OpenUI/*CloseUI correctly should be able to parse for the new features in the same manner. This enables a print manager to add new items to the user interface without the print manager itself having to be rewritten.

***OrderDependency:** *real section mainKeyword optionKeyword*

This keyword classifies and provides a partial ordering of the fragments of device-specific code in a PPD file. This allows a print manager to output the code in the correct order, so that later fragments will not undo the effects of earlier fragments. See section 2.5 for more information about order dependency.

The values are all StringValues. The *mainKeyword* and *optionKeyword* parameters specify a piece of device-specific code present in the PPD file. The *optionKeyword* must be omitted if the feature in question has no option keyword. The *optionKeyword* may be omitted if the code fragments for all option keywords for the given feature have the same classification and ordering.

The *section* parameter describes where in the job the code fragment can be emitted. The possible values for *section* correspond to the sections of a document file, as defined by the *Adobe Document Structuring Conventions, version 3.0*. Valid values for *section* are as follows:

- ExitServer—This code makes a change to the device which will take effect at the start of the next job. This code should be sent as a separate job, before the job which it will affect. The correct password (the value of *Password) must be supplied before this invocation.

- Prolog—This code supplies resources that must be in the Prolog section of a document.

- DocumentSetup—This code must be emitted in the Document Setup section, after the %%BeginSetup comment

- PageSetup—This code must be emitted in the Page Setup section after the %%BeginPageSetup comment and before the page level **save**.

- JCLSetup—This code provides job level control for devices that support a job control language. This code must be emitted after the code emitted by *JCLBegin and before the code emitted by *JCLToPSInterpreter.

- AnySetup—This code must be emitted in either the document setup or page setup section.

The parameter *real* specifies the relative order in which this code should be emitted. It defines the ordering across all device-specific code fragments, not just the code in the same *section*. The absolute values of ordering numbers are not important, only their values relative to other ordering numbers. Code assigned a lower number should be emitted before code assigned a higher number. Multiple code fragments can be assigned the same ordering number. If the numbers assigned to two code fragments are equal, they can be emitted in either order.

Some code fragments might not have an *OrderDependency entry, and thus are not assigned an ordering number or a section. Such code fragments can be used anywhere during the imaging of a page. Executing such code will affect the appearance of future imaging, but will not affect imaging already done. Such code has no defined order, except that it must be emitted after all fragments that do have ordering numbers.

The following example illustrates that code to change the resolution must be emitted before code to change the input slot:

```
*OpenUI *Resolution: PickOne
*OrderDependency: 10 AnySetup *Resolution
*Resolution 2504: "..."
*CloseUI: *Resolution

*OpenUI *InputSlot: PickOne
*OrderDependency: 20 AnySetup *InputSlot
*InputSlot Upper: "..."
*InputSlot Lower: "..."
*CloseUI: *InputSlot
```

The *OrderDependency entry for a given main keyword should be within the *OpenUI/*CloseUI bracketing for that keyword.

***UIConstraints:** *keyword1 option1 keyword2 option2*

This keyword, whose values are StringValues, denotes a mutually exclusive set of keywords representing device features. It tells a print manager which device features cannot be supported together, allowing the print manager to prevent the selection of both features at once in the user interface. For example, a device might forbid duplex printing when feeding from the envelope tray, or perhaps A4 size paper can only be fed from the upper slot, which would be expressed as a constraint of A4 page size and all input slots other than the upper slot.

This keyword can also be used to express the relationship between an optional component and the features it enables. For example, the presence of a fax board would enable the optional fax feature, or the presence of an extra memory board might enable higher resolution settings, but the absence of these optional components would prevent the display and use of their associated features.

The syntax is that the first keyword-option pair invalidates the second keyword-option pair. That is, if the first keyword-option pair is invoked, the device will not allow the second keyword-option pair to be invoked. Constraints on a device are usually reciprocal, so there will usually be two entries for each pair of features. One entry tells you that FeatureA-Option1 constrains FeatureB-Option5, and the other entry tells you that FeatureB-Option5 constrains FeatureA-Option1.

For example, a device might not allow transparencies to be output into the upper output bin, because transparency stock is too stiff to go through the convoluted paper path leading to the upper output bin. The *UIConstraints entry would look like this:

```
*UIConstraints: *MediaType Transparent  *OutputBin Upper
*UIConstraints: *OutputBin Upper *MediaType Transparent
```

The semantics of this in a user interface might be as follows: When the media type Transparent has been selected, the output bin labeled Upper is not available. The reverse is also true; when the output bin labeled Upper has been selected, the media type Transparent is not available. A print manager should make the choices unavailable to the user, either by “graying out” these items in the print panel or by some other method appropriate to the user interface.

The second keyword-option pair can also be a keyword-value pair, if the keyword has no options. This is most commonly used when the presence of an optional hardware component is necessary to enable a feature. See Keyword-Value Pairs on page 62 for an example of this usage.

Alternatively, each feature or both features may be specified without any options. For example, a constraint might take this form:

```
*UIConstraints: *FeatureA *FeatureB Option1
```

If no option is specified for Feature A, then Feature B is unconstrained until some option of the first feature, other than None or False, is selected. At that point, Option 1 of Feature B becomes constrained. For example:

```
*UIConstraints: *Staple *MediaType Transparent
```

The semantics of this are as follows: If any option for *Staple, other than None or False, is selected, transparent media cannot be selected.

The complement of the above form is

```
*UIConstraints: *FeatureA Option1 *FeatureB
```

If no option is specified for Feature B, and Option 1 of Feature A is selected, the constraint forces the selection of the None or False option of Feature B. For example:

```
*UIConstraints: *MediaType Transparent *Staple
```

If transparent media is selected, no *Staple options can be selected, except for None or False.

Finally, it is legal to omit options for both Feature A and Feature B. This means that if the option selected for Feature A is None or False, Feature B is unconstrained, but if any other option of Feature A is selected, the only valid options for Feature B are None or False.

It is illegal to omit an option for Feature B if it is a PickOne style without a None option, because that effectively disables all options of Feature B. It is also illegal to omit an option for Feature A if it is a  style without a None style, because it effectively disables the specified option of Feature B for all cases.

***Include:** *"filename"*

This allows the explicit inclusion of another PPD file (or partial file) into the current PPD file. The QuotedValue *filename* is the name of the file to be included. See section 4.2 for details on the syntax of *filename*.

*End

This keyword is used to close multiple-line InvocationValues and QuotedValues. The double quotes are still used to delimit the code sequence, but as an extra parsing check and for added human readability, the *End keyword is included after the closing double quote of a multiple-line PostScript language sequence. The keyword itself is of type NoValue.

## 5.6 Symbolic References to Data

The keywords in this section provide a way for parsers to skip large amounts of data contained in a PPD file when the parsers are not interested in that particular type of data. This is accomplished by providing a symbolic reference in place of a large body of PostScript language code.

Where an InvocationValue is normally permitted, it is legal to have a symbol name instead. A symbol name must start with the character caret, ^ (decimal ASCII 94). This symbol name is associated with a PostScript language sequence (InvocationValue) that appears at some later place in the PPD file (or in an attached local customization file). Since the InvocationValue might be large, a length hint can be provided to allow parsers to skip the large value quickly.

For example:

```
*OpenUI *MainFeature: PickOne
*MainFeature Option1: ^MySymbol
*MainFeature Option2: "..."
*CloseUI: *MainFeature
...
*SymbolLength ^MySymbol: bytecount EOLcount
*SymbolValue ^MySymbol: "
  ... bulky data here (e.g. color rendering dictionary)
"
*SymbolEnd: ^MySymbol
```

If a parser encounters a symbol name as a value, the parser should expect to find a *SymbolValue entry with the same symbol name later in the file. The rest of this section describes the individual keywords used to indicate symbolic pointers to bodies of data.

*Note* *The use of a symbol name in place of an InvocationValue is incompatible with version 3.0 of the PPD file specification and might cause problems for existing parsers. It is intended for use only when the code is very large—on the order of tens of kilobytes—and this is likely to occur only with new feature keywords which existing parsers are not scanning. Symbol names should not be used as the values of keywords that existed in older versions of this specification or for commonly-referenced keywords, such as *PageSize.*

***SymbolLength** *symbolName*: *bytecount EOLcount*

This keyword provides a hint to a parser about how long the body of data is, so the parser knows how many bytes to skip if it wants to skip this data. The option *symbolName* must be the same as the *symbolName* used in the associated *SymbolValue entry.

The StringValue values *bytecount* and *EOLcount* are unsigned integers. Together they measure the length in bytes of the data from the '*' byte of *SymbolValue to the '*' byte of *SymbolEnd. The first value gives the number of bytes, excluding the bytes that comprise end-of-line sequences. The second value gives the number of end-of-line sequences.

The parser must determine the number of bytes in an end-of-line sequence in the PPD file (usually 1 or 2).This number is usually a function of the operating system or platform on which the parser is operating, so it is usually known to the parser. It can then compute the byte offset of the *SymbolEnd keyword in the file by the formula

```
ibEnd = ibValue + bytecount + (cbEOL*EOLcount)
```

where

```
ibEnd   = byte offset of '*' in '*SymbolEnd'
ibValue = byte offset of '*' in '*SymbolValue'
cbEOL   = number of bytes per end-of-line sequence
```

and the values of *bytecount* and *EOLcount* are taken from the *SymbolLength keyword.

The information given by the *SymbolLength keyword is a hint only; parsers must not rely on it being correct or even present. If it is not correct or present, the parser must skip the value definition by scanning through the file until it reaches the *SymbolEnd keyword with the appropriate *symbolName*.

*SymbolLength must occur in a PPD file immediately before *SymbolValue.

***SymbolValue** *symbolName*: *"invocation"*

This keyword marks the beginning of a body of data of type InvocationValue. Symbol names must be defined in a *SymbolValue entry if they are referenced by a main keyword. It is an error for a PPD file to reference a symbol name that is not later defined. If a name is referenced but not defined, parsers can substitute a value of " " (empty quotes). The *SymbolValue entry for a given *symbolName* must occur after the reference to *symbolName* by a main keyword. That is, once a parser encounters a main keyword referencing *symbolName*, it can expect to find a corresponding *SymbolValue entry either later in the PPD file or in an included PPD file. The following two examples are both valid.

**Example 1:** *PPD file 'A'*

```
*%File: A
...
*Jog True: ^JogTrue
...
*SymbolLength ^JogTrue: 2000 500
*SymbolValue ^JogTrue: "..."
*SymbolEnd: ^JogTrue
...
```

**Example 2:** *PPD file 'B'*

```
*%File: B
...
*Jog True: ^JogTrue
...
*Include: "C" --------->          *%File: C
                                  ...
...                               *SymbolLength ^JogTrue: 2000 500
                                  *SymbolValue ^JogTrue: "..."
                                  *SymbolEnd: ^JogTrue
```

Alternatively, the reference to *symbolName* can be in an included PPD file as long as the *****SymbolValue entry is encountered after the *Include statement in the including file. For example, the following is valid because included files are treated as in-line files, so the parser must finish parsing File E before it encounters the *****SymbolValue in File D

```
*%File: D
...
*Include: "E" ------------------->              *%File: E
...                                             *Jog True: ^JogTrue
...                                             ...
*SymbolLength ^JogTrue: 2000 500
*SymbolValue ^JogTrue: "..."
*SymbolEnd: ^JogTrue
```

The rules for legal *symbolName*s are the same as for legal option keywords. By convention, Adobe-generated PPD files will construct symbol names by concatenating the main and option keywords together (omitting the *asterisk* from the main keyword). For example:

```
*ColorRenderDict Bond: ^ColorRenderDictBond
...
*SymbolValue ^ColorRenderDictBond: “...”
```

If there are multiple occurrences of *SymbolValue for a given *symbolName*, the first occurrence has precedence.

***SymbolEnd:** *symbolName*

This keyword must appear on the next line following the closing *double quote* of the symbol value. The StringValue value, *symbolName*, must be the same as the *symbolName* used in the associated *SymbolValue entry.

## 5.7 Installable Options

Most printers ship in some standard, minimal configuration but accept optional features or accessories, usually sold separately. These *installable options* can be paper trays, envelope feeders, font cards, and so on. The PPD specification provides a way to describe these accessories, label them as optional and initially not installed, and a way to install them later. Thus an application can list the installable options in its user interface, but can display them in some special way (for instance, grayed out) to indicate that the basic configuration does not support them.

Additionally, a customization file can be created to reflect a specific printer, and within that customization file, certain accessories can be marked as installed, so that applications can then allow them to be selected from the

user interface. The PPD specification also offers a way for an application to query the user for this configuration information, which can be used to update an application's internal database.

Without this information, an application might display all installable options, whether they are installed or not, and risk having the user select an uninstalled option and get errors or unexpected results.

### Syntax and Use

The *OpenGroup structure keyword (described in section 5.5) is used to denote the beginning of the installable options group. The option keyword InstallableOptions describes this special group. InstallableOptions is a registered option keyword that should not be used as an option for any other group in a PPD file. Like other values, it can have a translation string attached for clarity.

For example:

```
*OpenGroup: InstallableOptions/Options Installed
```

The installable options group contains one entry for each optional accessory that the printer can accept. Each entry consists of an *OpenUI/*CloseUI keyword pair, which surrounds the choices for the accessory. Within the entry, the *Default- entry denotes the current state of the accessory; that is, whether it is installed or not.

The actual main keywords used in this group consist of the generic string *Option followed by an integer; for example, *Option1, *Option2. Each installable option (each *OpenUI entry) must have a unique *Option name. The *UIConstraints section then maps the generic *Option keywords to the actual PPD feature entries. For example:

```
*OpenGroup: InstallableOptions/Options Installed
*OpenUI *Option1/Envelope Feeder: Boolean
*DefaultOption1: False
*Option1 True/Installed: ""
*Option1 False/Not Installed: ""
*CloseUI: *Option1
*CloseGroup: InstallableOptions

*UIConstraints: *Option1 False *InputSlot Envelope
```

The *UIConstraints entry tells a print manager that if *Option1 is False, then the Envelope option of the *InputSlot keyword is not available for selection by the user.

A print manager can use the InstallableOptions group in two ways. First, at printer installation or configuration time, a print manager can create a configuration panel based on the information found in the InstallableOptions group.

On this configuration panel the print manager posts the optional accessories listed, using the PickOne and Boolean values of the *OpenUI entries to determine whether an accessory requires a menu of choices or a boolean check box that denotes whether or not an accessory is installed. The user then informs the print manager which printer accessories are installed by selecting from the menus or checking the check boxes for each optional accessory. The print manager then stores this information in an internal database and later uses it to decide which options to offer the user at print time.

Second, a user or application might permanently configure the print manager by providing a local customization file that contains *Default- settings that reflect the installation of accessories.

For example, a local customization file might contain the following:

```
*OpenUI *Option1/Envelope Feeder: Boolean
*DefaultOption1: True
*CloseUI: *Option1
```

From this entry, a print manager can record that the value of *Option1 is currently True and use that information, in conjunction with the *UIConstraints entry in the base PPD file, to later decide which other options to offer to the user at print time. If the author of the print manager does not want to offer a configuration panel that interacts with the user, the print manager can be coded so that it looks at the *Default- setting and treats it as if it were a selection from the user. Instead of querying the user for configuration information, the print manager relies on the *Default- settings to be correct. This method is perhaps simpler to implement, but is less flexible for the user and requires that the user or some application edit a local customization file to record the configuration information.

Most *OpenUI entries in the InstallableOptions group are Boolean choices, as shown in the examples above. If an entry contains mutually exclusive choices, those choices can be grouped into one entry. For example, if a printer accepts memory upgrades of 2 MB or 4MB, the user can add either 2 MB or 4 MB but not both. Therefore, this choice can be represented in one entry, like this:

```
*OpenGroup: InstallableOptions
*OpenUI *Option2/Memory Configuration: PickOne
*DefaultOption2: None
*Option2 None/Basic Memory: ""
*Option2 2Meg/2Meg Memory Upgrade: ""
*Option2 4Meg/4Meg Memory Upgrade: ""
*CloseUI: *Option2
*CloseGroup: InstallableOptions

*UIConstraints: *Option2 None *Smoothing Medium
*UIConstraints: *Option2 None *Smoothing Dark
*UIConstraints: *Option2 2Meg *Smoothing Dark
```

This *UIConstraints entry tells a print manager that if None has been selected for *Option2, then neither the Medium nor Dark options of the *Smoothing keyword are available, and if 2Meg has been selected for *Option2, then only the Dark option of *Smoothing is not available. This provides a way for the print manager to present various options to the user based on the amount of memory installed in the printer.

The InvocationValues of the main keywords are null quotes because no code is invoked; the print manager is simply recording information either from the user or from the *Default- entries.

### Keyword-Value Pairs

The constrained feature (the second keyword-option pair in the *UIConstraints entry) may also be a keyword-value pair, if the keyword has no options. For example, suppose a device could transmit faxes, but only if an optional fax board was installed. The keyword-value pair that denotes fax support is:

```
*FaxSupport: Base
```

The corresponding entry would look like this:

```
*OpenGroup: InstallableOptions
*OpenUI *Option5/Fax Board: Boolean
*DefaultOption5: False
*Option5 False/Not Installed: ""
*Option5 True/Installed: ""
*CloseUI: *Option5
*CloseGroup: InstallableOptions
*UIConstraints: *Option5 False *FaxSupport Base
```

If the constrained feature is a multi-valued keyword, additional *UIConstraints entries would be added for each value of the keyword. For example, if the *FaxSupport entry in the PPD file had multiple values:

```
*FaxSupport: Base Extended
```

then the corresponding *UIConstraints entries would be:

```
*UIConstraints: *Option5 False *FaxSupport Base
*UIConstraints: *Option5 False *FaxSupport Extended
```

***VMOption** *vmOption*: *"int"*

This QuotedValue denotes potential values of the *FreeVM keyword with various optional memory (VM) configurations installed. The *vmOption* None denotes the basic, standard memory configuration, with no additional memory upgrades.

A PPD file with a *VMOption entry must also have a *FreeVM entry. The *FreeVM entry gives the amount of memory available for use by a PostScript language job available in the product's standard configuration. At least one *VMOption entry will contain the same value as the *FreeVM entry.

For example, the following entry indicates that the standard configuration contains 100,000 bytes of free memory at boot time, while the upgrade called 2Meg provides 1,100,000 bytes of free memory:

```
*FreeVM: "100000"
*VMOption None/Standard: "100000"
*VMOption 2Meg: "1100000"
```

*UIConstraints, the InstallableOptions group, and *VMOption can be used to show how much free memory is made available by installing additional memory. In the following example, the printer in its standard configuration, with no memory upgrades, has 100,000 bytes of free memory. Memory upgrades to 2 megabytes and 4 megabytes are available, but are not installed. Each memory upgrade makes the appropriate *VMOption available.

```
*FreeVM: "100000"
*VMOption None/Standard: "100000
*VMOption 2Meg: "1100000"
*VMOption 4Meg: "3100000"
...
*OpenGroup: InstallableOptions
*OpenUI *Option2/Memory Configuration: PickOne
*DefaultOption2: None
*Option2 None/Basic Memory: ""
*Option2 2Meg/2Meg Memory Upgrade: ""
*Option2 4Meg/4Meg Memory Upgrade: ""
*CloseUI: *Option2
*CloseGroup: InstallableOptions

*UIConstraints: *Option2 None *VMOption
*UIConstraints: *Option2 2Meg *VMOption 4Meg
*UIConstraints: *Option2 2Meg *VMOption None
*UIConstraints: *Option2 4Meg *VMOption 2Meg
*UIConstraints: *Option2 4Meg *VMOption None
```

## 5.8 Introduction to Media Handling

PPD files are most commonly used to take advantage of the different media sizes supported by a device. There are many devices on the market and many different sizes and types of media and finishing features supported on each device. The actual invocation code for a particular type of media often varies from one device to another—it might require the use of the operator **setpageparams** on one device, **setpapertray** on another, and **setpagedevice** on a third. The keywords in the next several sections are used to address the issues of choosing the input media, selecting a method of output, and requesting various finishing features.

In many instances what is wanted, at a user level, is “please print this on ledger paper,” where the user does not care from which tray the paper comes. For this situation, there is a keyword, *PageSize, whose corresponding invocation code selects a tray that contains the requested size of paper. Unless there are special media handling needs, it is suggested that the *PageSize keyword be used to set an input media type.

For more control over the media handling capabilities, there are keywords for directly selecting the input slots, the output bins, the output order of the pages, the imageable area of a given page, and finishing features, such as stapling. Each of these has a specific use that might be needed beyond the simplistic notion “please give me ledger paper.” For instance, if the manual feed feature is used, the *PageRegion keyword should be used to set up an imageable area for the manually fed sheet of paper.

*Note* *The author of a print manager should assume that all media handling requests (requests for a particular page size, media tray, and so on) will initiate a new, blank page. That is, assume that a request for a media handling feature will clear the frame buffer and perform the equivalent of the PostScript language operators* ***initgraphics*** *and* ***erasepage****.*

*This does not happen on all PostScript Level 1 devices, but is true for all PostScript Level 2 devices, and to be safe, you should assume it will happen on all Level 1 devices. Print managers should ensure that all media handling requests are placed in the output file before any page manipulation is performed and before any marks are made on the page.*

A primary use of a PPD file is for a print manager to be able to determine a list of all supported media types and to be able to determine the salient features of each page size (for example, the media dimensions and the imageable area). This list can then be displayed to the user in a user interface, or consulted by the print manger when a user requests a certain page size. In addition to the keywords that supply invocation code for the various media types, there are keywords that provide information about each media size. For example, the physical media dimensions are described by the *PaperDimension keyword, and the actual area of the page which is “writable” by the PostScript interpreter is described by the *ImageableArea keyword.

## 5.9 Media Option Keywords

In a PPD file, each type of media is described by an option keyword. The same option keyword is used with several different main keywords to describe different aspects of a given media type. For example, the entries *PageSize Letter, *PaperDimension Letter, and *ImageableArea Letter all address different aspects of a letter-size page. Because the same option keywords are used often in the next several sections, they are all described in this introductory section. Any of these option keywords can be substituted where the placeholder word *mediaOption* appears in a main keyword description.

Note that other media option keywords can be added to the list of registered option keywords at any time. To ensure that the set of features is not artificially limited, a parser should parse for the complete list of option keywords in a main keyword entry delimited by *OpenUI/*CloseUI rather than parsing for specific option keywords. See section 5.5 for more information about the extensibility of option keywords using the *OpenUI/*CloseUI delimiters.

Remember that not all known media sizes are specified here, only the most common. A device manufacturer is free to request a new size to be listed in the PPD file for a new device. However, when Adobe is creating the PPD file, care will be taken to avoid duplicating the semantics of a registered option with a new option of a different name. Also, when creating a new option keyword, the capitalization conventions shown in the following tables will be followed as much as possible; that is, the first letters of logical words should be capitalized.

Each of these option keywords can be further qualified by an extension to indicate a slightly distinct treatment of the media size. The only currently defined qualifier is Transverse, which indicates that the paper is fed in an orientation that is rotated 90 degrees from the orientation of the base paper size. Since most printers feed paper with the short edge of the paper perpendicular to the feed direction (“short edge feed”), Transverse generally means “long edge feed.” That is, a Transverse size generally indicates that the long edge of the image (on an imagesetter) or of the physical page (on a cut-sheet printer) is perpendicular to the feed direction. The option keyword qualifiers are appended with a period, like this: Letter.Transverse, A5.Transverse, and so on.

*Note* *Feeding the media transversely does not affect the relationship of user space to the physical page, but it does mean that the page is oriented differently with respect to device space. Transverse is* ***not*** *the same as landscape orientation of a page. The orientation of the image on the page is exactly the same for transverse as for non-transverse pages. A page fed transversely will appear identical to a page fed non-transversely, except that on certain printers, certain patterns and asymmetric halftone screens will image differently*

*when the page is fed transversely, due to device and driver limitations. Also, on older devices, the printing speed when using the* ***image*** *operator might be different for a page that is fed transversely.*

Any of the media option keywords can have a serialization extension which is a number used to distinguish between two otherwise equivalent instances of the same option keyword. For example, if there are two Letter-size media trays, they can be numbered to differentiate them (as in Letter.1, Letter.2). These extensions can be combined with Transverse (for example, Letter.1.Transverse) or other qualifiers, as appropriate.

The option keywords describing page sizes are shown in a series of tables. The dimensions given below are in PostScript default units, and they refer to the actual physical dimensions of the page, not the imageable region, unless otherwise specified. When media sizes are given as ISO or JIS standard sizes, the dimensions in millimeters (mm) are exact, and the dimensions in PostScript default units and inches are approximate.

### ISO/JIS Standard “A” Sizes

The following is a table of option keywords for the standard ISO/JIS “A” paper sizes. All sizes are given with the *x* dimension first, followed by the *y* dimension.

**Table 2** *Option keywords for standard ISO/JIS “A” paper sizes*

| *Option Keyword* | *Size (points)* | *Size (mm)* | *Size (inches)* |
|---|---|---|---|
| **A0** | 2384 x 3370 | 841 x 1189 | 33.11 x 46.81 |
| **A1** | 1684 x 2384 | 594 x 841 | 23.39 x 33.11 |
| **A2** | 1191 x 1684 | 420 x 594 | 16.54 x 23.39 |
| **A3** | 842 x 1191 | 297 x 420 | 11.69 x 16.54 |
| **A4** | 595 x 842 | 210 x 297 | 8.27 x 11.69 |
| **A5** | 420 x 595 | 148 x 210 | 5.83 x 8.27 |
| **A6** | 297 x 420 | 105 x 148 | 4.13 x 5.83 |
| **A7** | 210 x 297 | 74 x 105 | 2.91 x 4.13 |
| **A8** | 148 x 210 | 52 x 74 | 2.05 x 2.91 |
| **A9** | 105 x 148 | 37 x 52 | 1.46 x 2.05 |
| **A10** | 73 x 105 | 26 x 37 | 1.02 x 1.46 |

### JIS Standard “B” Sizes

The following is a table of keywords for the standard JIS “B” paper sizes. All sizes are given with the *x* dimension first, followed by the *y* dimension.

**Table 3** *Option keywords for standard JIS “B” paper sizes*

| *Option Keyword* | *Size (points)* | *Size (mm)* | *Size (inches)* |
|---|---|---|---|
| **B0** | 2920 x 4127 | 1030 x 1456 | 40.55 x 57.32 |
| **B1** | 2064 x 2920 | 728 x 1030 | 28.66 x 40.55 |
| **B2** | 1460 x 2064 | 515 x 728 | 20.28 x 28.66 |
| **B3** | 1032 x 1460 | 364 x 515 | 14.33 x 20.28 |
| **B4** | 729 x 1032 | 257 x 364 | 10.12 x 14.33 |
| **B5** | 516 x 729 | 182 x 257 | 7.17 x 10.12 |
| **B6** | 363 x 516 | 128 x 182 | 5.04 x 7.17 |
| **B7** | 258 x 363 | 91 x 128 | 3.58 x 5.04 |
| **B8** | 181 x 258 | 64 x 91 | 2.52 x 3.58 |
| **B9** | 127 x 181 | 45 x 64 | 1.77 x 2.52 |
| **B10** | 91 x 127 | 32 x 45 | 1.26 x 1.77 |

### ISO Standard “B” Sizes

The following is a table of keywords for the standard ISO “B” paper sizes. All sizes are given with the *x* dimension first, followed by the *y* dimension.

**Table 4** *Option keywords for standard ISO “B” paper sizes*

| *Option Keyword* | *Size (points)* | *Size (mm)* | *Size (inches)* |
|---|---|---|---|
| **ISOB0** | 2835 x 4008 | 1000 x 1414 | 39.37 x 55.67 |
| **ISOB1** | 2004 x 2835 | 707 x 1000 | 27.83 x 39.37 |
| **ISOB2** | 1417 x 2004 | 500 x 707 | 19.68 x 27.83 |
| **ISOB3** | 1001 x 1417 | 353 x 500 | 13.90 x 19.68 |
| **ISOB4** | 709 x 1001 | 250 x 353 | 9.84 x 13.90 |
| **ISOB5** | 499 x 709 | 176 x 250 | 6.93 x 9.84 |
| **ISOB6** | 354 x 499 | 125 x 176 | 4.92 x 6.93 |
| **ISOB7** | 249 x 354 | 88 x 125 | 3.46 x 4.92 |
| **ISOB8** | 176 x 249 | 62 x 88 | 2.44 x 3.46 |
| **ISOB9** | 125 x 176 | 44 x 62 | 1.73 x 2.44 |
| **ISOB10** | 88 x 125 | 31 x 44 | 1.22 x 1.73 |

### ISO Standard “C” Envelope Sizes

The following is a table of keywords for the standard ISO “C” envelope sizes. All sizes are given with the *x* dimension first, followed by the *y* dimension.

**Table 5** *Option keywords for standard ISO “C” envelope sizes*

| *Option Keyword* | *Size (points)* | *Size (mm)* | *Size (inches)* |
|---|---|---|---|
| **C0** | 2599 x 3676 | 917 x 1297 | 36.10 x 51.06 |
| **C1** | 1837 x 2599 | 648 x 917 | 25.51 x 36.10 |
| **C2** | 1298 x 1837 | 458 x 648 | 18.03 x 25.51 |
| **C3** | 918 x 1296 | 324 x 458 | 12.75 x 18.03 |
| **C4** | 649 x 918 | 229 x 324 | 9.02 x 12.75 |
| **C5** | 459 x 649 | 162 x 229 | 6.38 x 9.02 |
| **C6** | 323 x 459 | 114 x 162 | 4.49 x 6.38 |
| **C7** | 230 x 323 | 81 x 114 | 3.19 x 4.49 |
| **DL** | 312 x 624 | 110 x 220 | 4.33 x 8.66 |

### Other Standard Page Sizes

The following option keywords include U.S. standard paper and envelope sizes and other sizes that printer manufacturers have created and found useful. Unlike the ISO and JIS standard sizes, these U.S. standard sizes are typically defined in inches, and the millimeter measurements are provided only for comparison with the ISO and JIS standard sizes. All sizes are given with the *x* dimension first, followed by the *y* dimension.

**Table 6** *Other standard page sizes*

| *Option Keyword* | *Size (points)* | *Size (mm)* | *Size (inches)* |
|---|---|---|---|
| **Letter** | 612 x 792 | 215.9 x 279.4 | 8.5 x 11 |
| **Legal** | 612 x 1008 | 215.9 x 355.6 | 8.5 x 14 |
| **Statement** | 396 x 612 | 139.7 x 215.9 | 5.5 x 8.5 |
| **Tabloid** | 792 x 1224 | 279.4 x 431.8 | 11 x 17 |
| **Ledger** | 1224 x 792 | 431.8 x 279.4 | 17 x 11 |
| **7x9** | 504 x 648 | 177.8 x 228.6 | 7 x 9 |
| **9x11** | 648 x 792 | 228.6 x 279.4 | 9 x 11 |
| **9x12** | 648 x 864 | 228.6 x 304.8 | 9 x 12 |
| **10x13** | 720 x 936 | 254.0 x 330.2 | 10 x 13 |
| **10x14** | 720 x 1008 | 254.0 x 355.6 | 10 x 14 |

- LetterSmall—This is used to describe a reduced-size imageable region based on the Letter media size. Its imageable area varies across devices, but is generally about 552 x 730 points, centered on an 8.5 x 11 inch page.

- A4Small—This is used to describe a reduced-size imageable region based on the A4 media size. Its imageable area varies across devices, but is generally about 537 x 780 points, centered on an A4 page.

- Executive—This describes a page whose physical size is generally about 540 x 720 points (7.5 x 10 inches). If a device offers several “executive” sizes that are very similar (7.25 x 10.5 inch (522 x 756 points), 7.5 x 10.5 inch, 7 x 10.5 inch, and so on), these sizes can be differentiated by a serialization extension and a translation string that denotes the exact size. For example:

```
*PageSize Executive.1/7.5 x 10 in: "7.5x10inchtray"
*PageSize Executive.2/7.25 x 10.5 in: "7.25x10.5inchtray"
*PageSize Executive.3/7.5 x 10.5 in: "7.5x10.5inchtray"
```

- Folio—This is a 567 x 903.5 point imageable region, centered on an 8.5 x 13 inch (595 x 936 point) page (folio sheet).
- Quarto—This is a 567 x 744 point imageable region, centered on a 610 x 780 point page (quarto sheet).
- Comm10—This is a 297 x 684 point (4.125 x 9.5 inch) envelope.
- Monarch—This is a 279 x 540 point (3.875 x 7.5 inch) envelope.
- Envelope—This is a page size reserved for printing envelopes that have no standard name. This keyword can be qualified by an *x* and *y* dimension (specified in points), in the order **x.y**, where *x* is perpendicular to the feed direction and *y* is parallel to the feed direction.

*Note* *In the 3.0 specification, all envelope sizes were specified in the format Envelope.x.y, where x and y were the width and height of the envelope in PostScript default units, with the x dimension perpendicular to the feed direction. The option keyword could be followed by a translation string for clarity. For example, Envelope.612.792/Letter Envelope was a valid option. In the 4.0 and later specifications, envelopes are simply another size of media, and most envelope sizes will be listed by their common names (Comm10, Monarch), but the 3.0 format is still recognized and is useful for envelopes without common names.*

## 5.10 Media Selection

***PageSize** *mediaOption*: *"invocation"* UI

*Required*. This keyword provides the InvocationValue to invoke supported page sizes. The invocations in this section will establish both an input slot and a frame buffer (an area in device memory to hold the imageable region of the page). The exception to this is on roll-fed devices, such as imagesetters, where there are no selectable input slots and the invocation will only set up the frame buffer.

*PageSize should be used for the common case of a request for a certain size of media, with no special handling of media requested (for example, the user says, "give me legal size paper," but does not care which tray is used). *PageSize is intended to be used in all but very specific circumstances (such as when using manual feed or when directly controlling a media tray).

The invocation strings provided in the *PageSize section can provoke an error if the appropriate size of media is not available. The error message can be caught by the print manager and a more meaningful message generated.

An invocation string supplied by *PageSize will generally override an invocation string supplied by *PageRegion. Therefore, if a *PageRegion invocation is present in the output file, it must come after a *PageSize invocation to achieve the expected results. In a PPD file for an imagesetter, the invocation strings for *PageSize and *PageRegion are usually identical.

***DefaultPageSize:** *mediaOption*

This keyword indicates the default page size for the device when powered on. The value will correspond to one of the media options listed under *PageSize, or it will be Unknown. On devices that support more than one page size, the value will generally be Unknown, as it is impossible to predict which media tray will be inserted or designated as the default tray.

***?PageSize:** *"query"*

This query returns the media option corresponding to the current page size. The value returned must be one of the options listed under *PageSize or it will be Unknown.

***PageRegion** *mediaOption*: *"invocation"* UI

*Required*. These InvocationValues set the imageable area to the appropriate media type without explicitly specifying the source of the media. It is intended to be used in conjunction with manual feed so that the imageable area is appropriate for the media to be fed. It is also used instead of the *PageSize invocations when the user specifies an input tray and a page size (for example, Upper Tray, Letter Size), because the *PageSize invocations generally select an input tray and would override the user's previous selection of a specific input tray.

***DefaultPageRegion:** *mediaOption*

This indicates the default imageable area (in terms of media options) for the device when powered on. The value will correspond to one of the media options listed under *PageRegion or it will be Unknown. Like *DefaultPageSize, on most devices the value will be Unknown, because the imageable area is usually based on the size of the current media tray.

*Note* *The keywords *DefaultPaperTray, *PaperTray, and *?PaperTray, which were present in the 3.0 spec, have been removed because they were redundant. In their place, *PageSize should be used to select a particular size of paper, *PageRegion should be used to select a particular imageable area for manualfeed, and *InputSlot should be used to select a specific media tray. *InputSlot is documented in section 5.13.*

***MediaType** *typeOption*: "*invocation*"

UI

This keyword provides InvocationValues to select media by some characteristic other than size (or in addition to size). The options are product-dependent strings that describe the media. For example, a user might be able to select letterhead paper by specifying Letterhead as a media type. This method usually requires prior device setup, so that the device knows how to access a certain type of media.

***DefaultMediaType:** *typeOption*

This indicates the default media type. The value must match one of the options listed under *MediaType.

***?MediaType:** "*query*"

This query returns a string denoting the currently selected media type. The returned value must match one of the options under *MediaType or it will be Unknown.

***MediaColor** *colorOption*: "*invocation*"

UI

This keyword provides InvocationValues to select media by color. The options are product-dependent strings that describe the available colors of media. This method usually requires prior device setup, so that the device knows how to access a certain color of media.

***DefaultMediaColor:** *colorOption*

This indicates the default media color. The value must match one of the options listed under *MediaColor.

***?MediaColor:** "*query*"

This query returns a string denoting the currently selected media color. The returned value must match one of the options under *MediaColor or it will be Unknown.

***MediaWeight** *weightOption*: *"invocation"*

UI

This keyword provides InvocationValues to select media by weight. The options are product-dependent strings that describe the available media weights. This method of media selection usually requires prior device setup, so that the device knows how to access a certain weight of media

***DefaultMediaWeight:** *weightOption*

This indicates the default media weight. The value must match one of the options listed under *MediaWeight.

***?MediaWeight:** *"query"*

This query returns a string denoting the currently selected media weight. The returned value must match one of the options under *MediaWeight or it will be Unknown.

## 5.11 Information About Media Sizes

***ImageableArea** *mediaOption*: *"$ll_x$ $ll_y$ $ur_x$ $ur_y$"*

*Required*. This provides the bounding box of the imageable area for each given page size (*mediaOption*). The bounding box is given as four real numbers, representing the x and y coordinates of the lower left and upper right corners of the region, respectively, in the PostScript language default user space coordinate system. The *x* and *y* axes of a given page size correspond to the *x* and *y* axes of that page size in the *PaperDimension entry.

The imageable region is defined as the part of the page where marks can actually be made. On many devices, there are margins imposed by the media transport mechanism in the marking engine that might prevent marks from being made close to the edges of the media. The *ImageableArea entry will supply a region that represents a "reliable" area of the page in which marks can be made. This might exactly correspond to the *clipping path* set by the PostScript interpreter. The value is represented as an InvocationValue.

On some devices, the imageable area of a given page size varies as a result of the current resolution, amount of memory, the direction of paper feed, and other factors. For example, the imageable area of a Legal size page might be smaller at higher resolutions on a printer with variable resolution, or it might be shifted left or right depending on whether the page was fed long-edge-first or short-edge-first. In PPD files where the imageable area of a given page size can vary depending on other factors, the imageable area recorded for that page size will be the intersection of all possible imageable areas for that page size. While this means that the imageable area available in the current configuration might actually be larger than the imageable area shown in the PPD file, it at least guarantees that the available imageable area will not be smaller than that shown in the PPD file, and all marks made within the given imageable area will be visible.

***DefaultImageableArea:** *mediaOption*

This provides the default imageable area in terms of media options. This value must be one of the media options listed under *ImageableArea or it will be Unknown.

***?ImageableArea:** *"query"*

This query returns four real numbers representing the bounding box of the imageable area, as defined under *ImageableArea. Since it is virtually impossible to determine hardware restrictions from software polling, this query will usually return the default clipping region for the page size in effect. In general, it is better to use the values supplied in the *ImageableArea keyword section, since they can be adjusted by hand for particular hardware constraints.

***PaperDimension** *mediaOption: "real real"*

*Required*. This lists physical dimensions for a particular media size, independent of the imageable area of the page. There are only two numbers specified, which represent the *width* (in the *x* dimension) and *height* (in the *y* dimension) of the media, respectively, in PostScript default units. The *x* and *y* axes of a given page size correspond to the *x* and *y* axes of that page size in the *ImageableArea entry. The value is represented as an InvocationValue.

***DefaultPaperDimension:** *mediaOption*

This provides the default physical media dimension in terms of media options. This value must be one of the options listed under *PaperDimension or it will be Unknown.

***LandscapeOrientation:** Plus90 | Minus90 | Any

Every print manager makes assumptions about the location of the origin of default user space on the physical page. When a user selects landscape orientation, a print manager must rotate and translate the origin of default user space on the page. On certain printers, the orientation of the physical page is dictated by either physical markings on the printer case, or by instructions in the user manual. This dictated orientation might be incompatible with the print manager's assumptions about the orientation of the physical page. This is not significant for blank paper, but for pre-marked paper, such as letterhead, 3-hole-punched paper, or envelopes, the printed output might appear upside-down with respect to the letterhead, punch holes, envelope flap, or other pre-markings on the page.

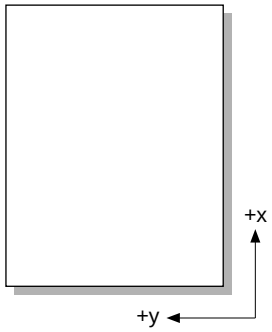
This keyword, whose value of type StringValue is determined from knowledge of the printer's markings and instructions, provides a hint to a print manager as to which way it should rotate and translate the page, for the printed output to be compatible with the page feeding instructions on the printer or in the printer's user manual. If this keyword is present, it means that the printer requires the use of the transformations listed below for the correct printing results to occur.

The values have the following meanings:

- Plus90—This means that the print manager should perform the functional equivalent of the following fragment of PostScript language code:

  ```
  90 rotate 0 pagewidth neg translate
  ```

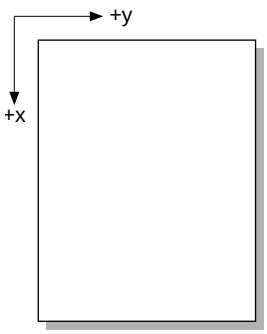
  where *pagewidth* is the width of the page in default user space. For example, on a letter-size page in portrait mode, after this transformation has been performed, the default user space would look like this:

- Minus90—This means that the print manager should perform the functional equivalent of the following fragment of PostScript language code:

  ```
  90 neg rotate pageheight neg 0 translate
  ```

  where *pageheight* is the height of the page in default user space. For example, on a letter-size page in portrait mode, after this transformation has been performed, the default user space would look like this:

- Any—This means that no hint is provided and the driver can follow its normal assumptions, but the results might be incorrect for certain printers.

*LandscapeOrientation should appear only in the PPD files of printers in which the orientation of page feeding is dictated by printer markings or the printer's user manual. If this keyword is missing, assume that Any is the default value.

*Note* *If a printer treats envelopes differently from paper (for example, when an envelope size is requested, the printer performs its own rotations and translations to print "correctly" on the envelope), this keyword might not provide any assistance and the printing results might still be incorrect.*

## 5.12 Custom Page Sizes

Some devices support user-defined page sizes by allowing the user to supply the page dimensions rather than selecting from a list of pre-defined page sizes. The keywords in this section are provided to support that capability.

Devices are divided into two categories, based on whether the device supports *roll-fed media* or *cut-sheet media*. On roll-fed media devices, such as an imagesetter supplied by a roll of film, the media is larger than the page size requested by the user. The requested page size is positioned somewhere on the larger physical media, and the imageable area is assumed to be identical to the requested page size.

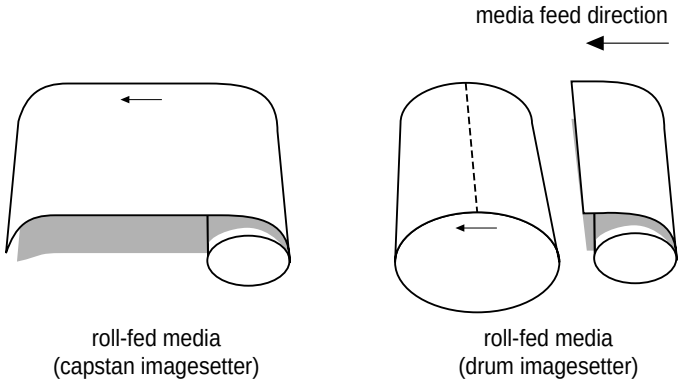
Cut-sheet media devices accept individual pages of physical media of different sizes. The user is expected to supply media of the correct physical size, usually in a tray that adjusts to different media sizes. The requested page size is identical to the physical page size. However, due to media handling hardware requirements, the imageable area may be smaller than the requested page size. The unimageable margin area required by the hardware is denoted by the keyword *HWMargins, which is described in this section.

The location and orientation of the page image on the media and of the initial PostScript coordinate system depends on the combination of the custom page size parameters. These parameters are named the same but are treated differently on roll-fed and on cut-sheet devices.

### Custom Page Size Parameters for Roll-Fed Devices

On a roll-fed device, custom page size and orientation parameters are specified relative to the *media feed direction*, which is parallel to the length of the roll of media. Figure 1 illustrates media feed direction on a roll-fed device:

**Figure 1** *Media feed direction*




Custom page sizes are defined in terms of the following parameters:

- Width—On a roll-fed device, this indicates the width of the page perpendicular to the direction of media feed, in PostScript default units.

- Height— On a roll-fed device, this indicates the height of the page parallel to the direction of media feed, in PostScript default units.

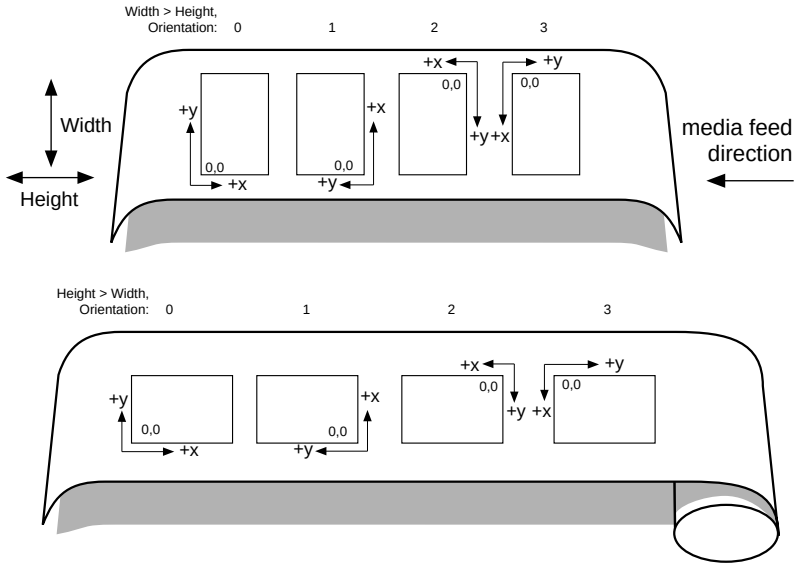
- WidthOffset—On a roll-fed device, this indicates the amount, in PostScript default units, to offset the image perpendicular to the direction of media feed. The direction of the offset is in the direction of increasing *y* in user space for orientation 0.

- HeightOffset—On a roll-fed device, this indicates the amount, in PostScript default units, to offset the image parallel to the direction of media feed. The direction of the offset is in the direction of increasing *x* in user space for orientation 0.

- Orientation—On a roll-fed device, this indicates the orientation of the image with respect to the media feed direction. Devices support a subset of four possible values. In orientation 0, the *x* axis in user space decreases in the media feed direction. The *y* axis therefore increases 90 degrees counterclockwise relative to increasing *x*, orthogonally to media feed direction. Orientations 1, 2, and 3 are rotated 90, 180, and 270 degrees (respectively) counterclockwise from orientation 0.

Figure 2 shows the interaction between Width, Height and Orientation for roll-fed media devices. Note that Width and Height are always defined with respect to the media feed direction. For a given Width and Height, two values of Orientation will produce a landscape coordinate system and two will produce a portrait coordinate system.

**Figure 2** *Roll-fed media devices*




*Note* *On both cut-sheet and roll-fed devices, the actual size of a page might not be exactly what was requested, according to the current configuration of the device. On a roll-fed device, the actual orientation of a page might not exactly match the request, again because of the device's configuration. For example, an imagesetter manufacturer might configure a product to rotate a page automatically if the page cannot fit in the requested long-edge feed orientation. The *CustomPageSize invocation code cannot be expected to override such behavior.*

### Custom Page Size Parameters for Cut-Sheet Devices

On a cut-sheet device, there is no relationship between the media feed direction and the custom page size parameters. The parameters are defined as follows:

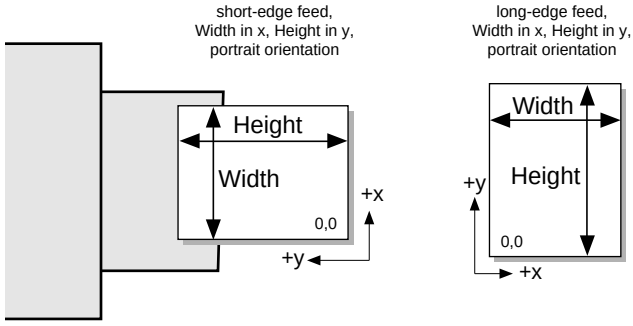
- Width—On a cut-sheet device, Width describes, in PostScript default units, the width of the physical page as measured along the *x* axis.
- Height— On a cut-sheet device, Height describes, in PostScript default units, the height of the physical page as measured along the *y* axis.
- WidthOffset—On cut-sheet devices, this parameter has no effect, so the *CustomPageSize code will discard it.

- HeightOffset—On cut-sheet devices, this parameter has no effect, so the *CustomPageSize code will discard it.

- Orientation—On cut-sheet devices, this parameter has no effect, so the *CustomPageSize code will discard it. If a print manager wants to offer landscape orientation to the user, it must provide its own rotation code and user interface.

Figure 3 shows the interaction between Width and Height on cut-sheet media devices. The image is assumed to be in portrait orientation. Note that Width and Height are constant relative to the *x* and *y* axes of the page respectively, even though the media is fed long-edge first in one case and short-edge first in the other case.

**Figure 3** *Cut-sheet media devices*




***CustomPageSize** True: "*invocation*"

This InvocationValue provides the code to set up a custom page size. The print manager is responsible for obtaining five parameters from the user and placing them on the operand stack in the correct order before executing the invocation code. The parameters are Width, Height, WidthOffset, HeightOffset, and Orientation, as described earlier. The order in which these parameters must be placed on the stack is described under *ParamCustomPageSize.

*Note* *To be compatible with existing parsers, *CustomPageSize conforms to the syntax of other True/False entries, but there is no reason to ever have a *CustomPageSize False entry, since there is no sensible corresponding invocation code.*

***ParamCustomPageSize** *paramOption*: *order type min max*

This provides the allowable types and ranges, expressed as a StringValue, for each of the custom page size parameters required by the invocation code of the *CustomPageSize entry. There must be one *ParamCustomPageSize entry for each of the custom page parameters: Width, Height, WidthOffset,

HeightOffset, and Orientation. Like any option keyword, these options can have translation strings, allowing a print manager a more meaningful string to display to the user.

The value of *order* indicates the order in which this parameter must be placed on the stack and passed to the *CustomPageSize code. A parameter with an order of "1" is placed on the stack first, followed by a parameter with an order of "2", and so on. An application program is responsible for obtaining these parameters from the user and putting them on the stack in the correct order before invoking the *CustomPageSize code.

The *type* of each parameter is either *int*, *real*, or *points*, where *points* is a real number of PostScript default units. A value of *points* tells an application that, although the units might be obtained from the user in any form, such as inches or millimeters, they must be translated to PostScript interpreter's default units (1/72 inch) before they are placed on the stack.

The allowable range for each parameter is expressed as the minimum and maximum acceptable numbers, inclusive, with the minimum value first. The type of the minimum and maximum acceptable values for a parameter must match the *type* entry for that parameter. A print manager should use the minimum and maximum values for each parameter to ensure that the user provides parameters in the valid range.

The following is a sample custom page size entry for a PostScript Level 1 roll-fed device:

```
*LanguageLevel: "1"
*CustomPageSize True:"exch pop% discard HeightOffset
  statusdict /setpageparams get exec"
*End
*ParamCustomPageSize Width: 1 points 1 792
*ParamCustomPageSize Height: 2 points 1 5184
*ParamCustomPageSize WidthOffset: 3 points 0 791
*ParamCustomPageSize HeightOffset: 4 points 0 0
*ParamCustomPageSize Orientation: 5 int 0 1
```

*Note* *On some devices, requesting a width or height of zero will cause an error. Because of this, the minimum width and height bounds are set to small positive numbers. On PostScript Level 1 roll-fed devices that use the* ***setpageparams*** *operator in the *CustomPageSize code, HeightOffset is not used, and thus will be discarded by the *CustomPageSize invocation code, as shown above.*

The following is a sample custom page size entry for a PostScript Level 2 roll-fed device:

```
*LanguageLevel: "2"
*CustomPageSize True: "
  4 dict begin
  dup /Orientation exch def
  /PageOffset [ 5 -2 roll ] def
  /PageSize [
    5 2 roll dup 0 eq exch 2 eq or {exch} if
  ] def
  /ImagingBBox null def
currentdict end setpagedevice
"
*End
*ParamCustomPageSize Width: 1 points 100 864
*ParamCustomPageSize Height: 2 points 100 5184
*ParamCustomPageSize WidthOffset: 3 points 0 764
*ParamCustomPageSize HeightOffset: 4 points 0 5084
*ParamCustomPageSize Orientation: 5 int 0 3
```

The following is a sample custom page size entry for a PostScript Level 2 cut-sheet device:

```
*LanguageLevel: "2"
*CustomPageSize True: "
   pop pop pop % discard orientation & offsets
   2 dict begin
   /PageSize [
     4 -2 roll
   ] def
   /ImagingBBox null def
currentdict end setpagedevice
"
*End
*ParamCustomPageSize Width: 1 points 100 864
*ParamCustomPageSize Height: 2 points 100 1008
*ParamCustomPageSize WidthOffset: 3 points 0 0
*ParamCustomPageSize HeightOffset: 4 points 0 0
*ParamCustomPageSize Orientation: 5 int 0 0
```

*Note* *On cut-sheet media devices, Orientation, HeightOffset and WidthOffset are not used, and thus are discarded by the *CustomPageSize invocation code.*

***MaxMediaWidth**: *"real"*

On devices that support custom page sizes, this QuotedValue entry indicates the maximum media width allowed by the device when a custom page size is requested. On both roll-fed and cut-sheet devices, *MaxMediaWidth is measured perpendicular to the media feed direction and is expressed in PostScript default units.

On roll-fed devices, a print manager must ensure that the sum of the requested Width plus the requested WidthOffset does not exceed the value of *MaxMediaWidth. On cut-sheet devices, since the media feed direction and orientation of the image are unknown, a print manager must ensure that the larger of Width and Height is less than or equal to the larger of *MaxMediaWidth and *MaxMediaHeight. Likewise, a print manager must ensure that the smaller of Width and Height is less than or equal to the smaller of *MaxMediaWidth and *MaxMediaHeight.

***?CurrentMediaWidth:** "*query*"

The absolute maximum width of media supported by a device, as measured perpendicular to media feed direction, can be obtained from the value of *MaxMediaWidth. However, some devices support different sizes of media cassettes, so the current maximum width might be less than the absolute maximum width. This query returns a real number specifying the maximum width, in PostScript default units, of the currently installed media.

If this query is available, a print manager can use it to replace the value of *MaxMediaWidth in the print manager's internal data structures with the value returned by the query. The print manager can then proceed with range-checking as described under *MaxMediaWidth.

***MaxMediaHeight**: *"real"*

On devices that support custom page sizes, this QuotedValue entry indicates the maximum media height allowed by the device when a custom page size is requested. On both roll-fed and cut-sheet devices, *MaxMediaHeight is measured parallel to the media feed direction and is expressed in PostScript default units.

On roll-fed devices, a print manager must ensure that the sum of the requested Height plus the requested HeightOffset does not exceed the value of *MaxMediaHeight. On cut-sheet devices, since the media feed direction and orientation of the image are unknown, a print manager must ensure that the larger of Width and Height is less than or equal to the larger of *MaxMediaWidth and *MaxMediaHeight. Likewise, a print manager must ensure that the smaller of Width and Height is less than or equal to the smaller of *MaxMediaWidth and *MaxMediaHeight.

***?CurrentMediaHeight:** *"query"*

The absolute maximum height of media supported by a device, as measured parallel to media feed direction, can be obtained from the value of *MaxMediaHeight. However, some devices support different sizes of media

cassettes, so the current maximum height might be less than the absolute maximum height. This query returns a real number specifying the maximum height, in PostScript default units, of the currently installed media.

If this query is available, a print manager can use it to replace the value of *MaxMediaHeight in the print manager's internal data structures with the value returned by the query. The print manager can then proceed with range-checking as described under *MaxMediaHeight.

***HWMargins:** *left bottom right top*

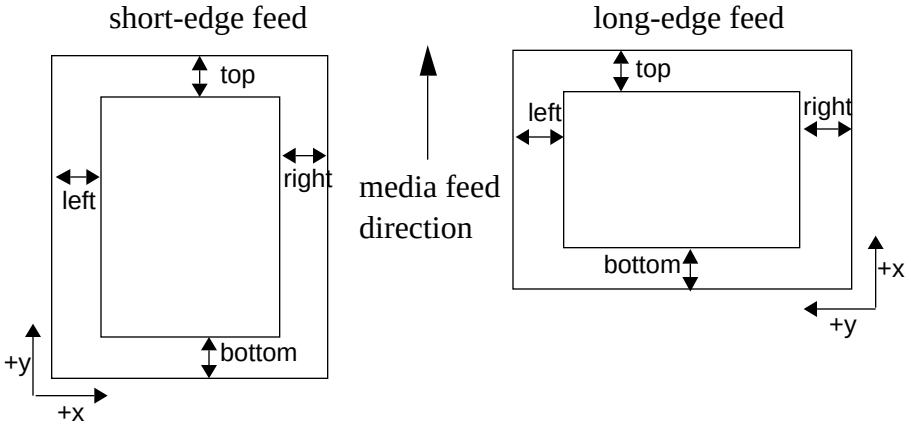
This keyword applies only to cut-sheet devices. It provides a StringValue that describes how much space around the outer edge of the page cannot be imaged because of hardware restrictions. A print manager might use this information to tell a user when the entire requested custom page size cannot be printed upon, or to show the user which part of the page can be imaged. For non-custom page sizes, the keyword *ImageableArea provides the same information for each supported page size.

The values of *HWMargins are in PostScript default units, and are defined in default user space as follows:

*top* = distance, in PostScript default units, from the top edge of the page to the nearest beginning of imageable area. The top edge is the edge of the page that enters the printer first.

*right*, *left*, and *bottom* are similarly defined, as shown in Figure 4.

**Figure 4** *Margins of *HWMargins*



For example, a printer might have an adjustable tray that accepts several sizes of paper, but the printer always needs 1 inch along the sides and 1/2 inch at the top and bottom to handle paper of any size. The *HWMargins entry to describe this would be

```
*HWMargins: 72 36 72 36
```

If the margin requirements of the printer vary with the paper size (for example, if the printer needs a 1 inch margin to handle some page sizes, and a 2 inch margin to handle other page sizes), the values of *HWMargins will reflect the largest margin required by the printer (in this case, the 2 inch margin). For some page sizes, this might provide a smaller imageable area than is actually achievable by the printer, but at least it guarantees that marks made within the indicated imageable area will be visible on the page.

Since the media feed direction on cut-sheet devices is not known, a print manager does not know which edge, long or short, is the top edge. Therefore. the print manager must pick the largest of the four *HWMargin values and subtract it from each edge of the page, to ensure a guaranteed imageable area.

*HWMargins applies only to cut-sheet media devices. Roll-fed media devices are assumed to be able to image over the complete requested page size; any restrictions on page size due to hardware control mechanisms are described by *MaxMediaWidth and *MaxMediaHeight. The presence of *HWMargins indicates a cut-sheet media device which supports *CustomPageSize. If such a device has no hardware restrictions on imageable area, it should still include the following entry to distinguish it from a roll-fed device that supports custom page sizes:

```
*HWMargins: 0 0 0 0
```

This keyword must *not* be present in the PPD file of a roll-fed media device or of a cut-sheet media device that does not support *CustomPageSize.

***CenterRegistered:** True | False

This keyword applies only to roll-fed devices. It provides a StringValue that tells whether the device registers the film or paper stock from the center or from the edge of the scan. If a device uses center-registering, it is up to the user or the application to provide the correct value for WidthOffset, to move the image to the beginning edge of the stock. For example, on a center-registered device, if the user requests 10-inch wide stock on a 12-inch wide transport mechanism, either the user or the application must provide a 1-inch WidthOffset to get the image to start at the edge of the stock. On a device that does not use center-registering, this additional calculation is unnecessary.

***VariablePaperSize:** True | False

This StringValue indicates whether the device supports infinitely variable media sizes.

*Note* **VariablePaperSize is retained only for backward compatibility with the 3.0 specification. The *CustomPageSize and *ParamCustomPageSize keywords handle custom media size more comprehensively and should be used instead.*

## 5.13 Media Handling Features

***InputSlot** *inputSlotOption*: *"invocation"* UI

This provides the InvocationValue to select input slots for media trays. If present, this keyword implies that media can be selected by specifying, for instance, the upper or the lower slot, and accepting whatever is found there. The most common use of this keyword is to select a media tray that contains letterhead or other special paper. One entry is provided for each input slot.

***DefaultInputSlot:** *inputSlotOption*

This denotes the default input slot. The value must match one of the available input tray options listed under *InputSlot or it will be Unknown.

***?InputSlot:** *"query"*

This query returns a string denoting the currently chosen input slot. The value returned must match one of the options listed under *InputSlot or it will be Unknown.

Any arbitrary strings that appropriately describe the printer's input slots are valid *inputSlotOption*s. The following list documents commonly used *inputSlotOption*s:

- Lower—This is used for any tray for which there is no particular distinguishing feature other than it is lower than another tray similar to it. For instance, if there are two media input slots that are identical, they can be distinguished by their position relative to each other.
- Middle—This designates a tray that is in between other trays. See Lower.
- Upper—This designates a tray that is above other trays. See Lower.
- Rear—This designates a tray at the rear of the printer.

- Cassette—This keyword can be used where Upper, Middle, and Lower make little sense (that is, if there is only one input slot, or if the printer is a roll-fed device). Since many print managers display the choices of input slot and manual feed on a single menu, Cassette provides differentiation for the user between the paper or film cassette and the manual feed slot, if one exists.

- LargeCapacity—This is used to refer to a large capacity media tray, such as an input paper tray that can hold more than one ream of paper.

- LargeFormat—This refers to an input slot that can hold "large format" media trays (ledger paper, for instance).

- AnySmallFormat—This is used to indicate a media tray that can hold any of the smaller format medias. This includes any media size that is up to (and including) 11 inches on the longer side.

- AnyLargeFormat—This option allows selection of a "universal" media tray that can contain any of the large format media sizes (those with one dimension greater than 11 inches).

***RequiresPageRegion** *inputSlotOption*: True | False

This keyword provides a StringValue that indicates, for each input slot, whether or not the page size installed in that slot can be sensed by the printer. If the page size cannot be sensed, any invocation of that input slot must be followed by an invocation of the appropriate *PageRegion code to set up the requested frame buffer and imageable area for the page. Therefore, if the printer cannot sense the page size in a slot, the *PageRegion code is required, and the value of *RequiresPageRegion will be True for that slot. If the page size in that slot can be sensed, the *PageRegion code is not required, and the value of *RequiresPageRegion will be False.

The option keyword *inputSlotOption* must be a valid *inputSlotOption* listed in the *InputSlot entry in the PPD file. For example:

```
*InputSlot Lower: "code"
*InputSlot Envelope: "code"
...
*RequiresPageRegion Lower: False
*RequiresPageRegion Envelope: True
```

An additional special option keyword All means that the entry applies to all paper sources on the printer. For example:

```
*RequiresPageRegion All: False
```

This entry indicates that all the input slots can sense the page size in them, and the *PageRegion code is never required after an input slot invocation.

If *RequiresPageRegion for any slot (or all slots) is omitted from a PPD file, it is assumed to be False for those slots. That is, the *PageRegion code should not be invoked after an input slot invocation.

***OutputBin** *binOption*: *"invocation"* UI

This provides the InvocationValue to select different output paths for media. If this entry is absent, this device does not provide software-selectable output paths.

***DefaultOutputBin:** *binOption*

This denotes the default output path. The value must match one of the available output paths listed under *OutputBin or it will be Unknown.

***?OutputBin:** *"query"*

This query returns a string denoting the current output path. The string returned must match one of the option keywords listed under *OutputBin or it will be Unknown.

The currently registered values for *binOption* are:

- Upper—This refers to an output bin located above any other output bins.
- Lower—This refers to an output bin located below any other output bins.
- Rear—This designates an output bin located to the rear of the device.
- OnlyOne—This keyword is used where Upper, Rear, and other positional keywords make little sense (that is, if there is only one output bin).

***OutputOrder** *orderOption*: *"invocation"* UI

This entry provides the InvocationValue to invoke a specific page stacking order for the duration of the current job. On many devices, the output order is tied to the selection of the output bin. On some devices, invoking a new page stacking order will cause a new output bin to be selected. On other devices, a new output bin must be explicitly selected.

***DefaultOutputOrder:** *orderOption*

This entry indicates the default page stacking order (in other words, the page stacking order of the default output bin). This entry is useful in determining in which order the pages should be sent from the host. The value must match one of the options listed under *OutputOrder.

***?OutputOrder:** *"query"*

This query returns a string denoting the page stacking order of the current output bin. The returned value must match one of the option keywords listed under *OutputOrder.

The currently registered values for *orderOption* are:

- Normal—This keyword indicates that if the pages are transmitted to the device in *1-n* order, they will be in *1-n* order when they are picked up from the output tray. This usually, but not always, means that the output pages are stacked face down in the output tray.
- Reverse—This keyword indicates that if the pages are transmitted to the device in *1-n* order, they will be in *n-1* order when they are picked up from the output tray (the last page will be on the top of the stack). This usually, but not always, means that the output pages are stacked face up in the output tray.

***PageStackOrder** *binOption*: Normal | Reverse

This is an informational entry with a value of type StringValue, which indicates the page stacking order of each output bin, for a device with multiple software-selectable output bins. The option keyword *binOption* must be a valid option keyword listed under *OutputBin. Normal and Reverse have the same meaning as defined under *OutputOrder.

There is an implicit assumption that the stacking order of a given bin cannot be changed. This entry is useful in determining either which output path to select (to get the proper page ordering) or in which order the pages should be sent from the host (to utilize the stacking order of the chosen output path). This keyword will not be present if the device has only one output bin.

***TraySwitch** True | False: *"invocation"* UI

This provides the InvocationValue to turn automatic tray switching on (True) and off (False). Automatic tray switching is provided by some devices with multiple input trays, so that when one input tray runs out of media, another tray with the same type of media can be automatically used.

***DefaultTraySwitch:** True | False

This denotes the default state of the automatic tray switching mechanism.

***?TraySwitch:** *"query"*

This query returns True if automatic tray switching is currently enabled, and False if it is currently disabled.

***ManualFeed** True | False: *"invocation"* UI

This provides the InvocationValue to turn manual feed on (True) and off (False).

***DefaultManualFeed:** True | False

This denotes the default state of the manualfeed mechanism. True means that manualfeed is enabled; False means that manualfeed is disabled.

***?ManualFeed:** *"query"*

This query returns True if manualfeed is currently enabled, or False if manualfeed is currently disabled.

***Signature** *signatureOption*: *"invocation"* UI

This provides the InvocationValue to invoke signature options. Signaturing is the automatic ordering of virtual pages on physical pages, so that the output, when properly folded and collated, will have all the virtual pages in the proper order. One of the options must be None or False, to turn off the automatic signature feature. Option keywords might include the number of virtual images per physical page.

The currently registered values for *signatureOption* are:

- True—Turn on the signature option.
- False—Turn off the signature option.

***DefaultSignature:** *signatureOption*

This denotes the default state of the automatic signature feature. The value must be one of the options listed under *Signature or it will be Unknown.

***?Signature:** "*query*"

This returns a string denoting the current state of the automatic signature feature. The value returned must be one of the options listed under *Signature or it will be Unknown.

***Duplex** *duplexOption*: *"invocation"* UI

This keyword provides the InvocationValue to control the duplex (two-sided printing) feature. One of the options must be None or False, for "no duplexing" (that is, produce simplex or one-sided printing).

***DefaultDuplex:** *duplexOption*

This denotes the default state of the duplex feature. The value must be one of the options listed under *Duplex or it will be Unknown.

***?Duplex:** *"query"*

This query returns a string denoting the current state of the duplexing mechanism. The value returned must be one of the option keywords listed under *Duplex or it will return Unknown.

These are the current option keywords used by the *Duplex keywords. Tumbling is defined in section 4.11 of the *PostScript Language Reference Manual, Second Edition*. Briefly, to print a book, where the binding is along the left edge, the user selects NoTumble. To print a calendar, bound along the top edge so that successive pages are flipped upward, the user selects Tumble. Tumble is also referred to as "HeadToToe."

- DuplexTumble—Print on both sides of the paper and tumble the images while printing.
- DuplexNoTumble—Print on both sides of the paper but do not tumble the images.
- SimplexTumble—Print on only one side of the paper, but tumble the images while printing.
- None—Print the image on one side of the paper and do not tumble successive images (this is "normal" one-sided printing, equivalent to SimplexNoTumble).

***OutputMode** *modeOption*: "*invocation*"

UI

This provides InvocationValue to invoke different output modes. Output modes might be caused by mechanical variations in the printer, such as varying print-head direction or speed. The valid values for *modeOption* are strings that describe the level of output quality (for example, Draft or LetterQuality).

***DefaultOutputMode:** *modeOption*

This denotes the default output mode. The value must be one of the options listed under *OutputMode or it will be Unknown.

***?OutputMode:** *"query"*

This query returns a string denoting the current output mode. The value returned must be one of the options listed under *OutputMode or it will be Unknown.

## 5.14 Finishing Features

This section documents finishing features, which typically affect a document after it has been printed or imaged. For the convenience of print managers, all finishing features in a PPD are grouped with *OpenGroup/*CloseGroup. For a complete example, refer to the sample PPD file in section 6.

***Collate** *collateOption: "invocation"* UI

This keyword provides the InvocationValue to control collating. Collating is defined as follows: for three copies of a three-page document, collated pages are produced in the order 1-2-3-1-2-3-1-2-3, while uncollated pages are produced in the order 1-1-1-2-2-2-3-3-3. One of the options must be None or False, to turn off collating.

The currently registered values for *collateOption* are

- True—Turn on collation.
- False—Turn off collation.

***DefaultCollate:** *collateOption*

This denotes the default state of the collator mechanism. The value must be one of the options listed under *Collate or it will be Unknown.

***?Collate:** *"query"*

This query returns a string denoting the current state of the collator mechanism. The value returned must be one of the options listed under *Collate or it will be Unknown.

***FoldType** *foldOption*: "*invocation*" UI

This keyword provides the InvocationValue to control which type of fold is invoked, if any. *FoldType should be ignored by a print manager unless *FoldWhen has been invoked with a value other than None.

The following are the current *foldOptions*. Many of these folds are illustrated by Figure G.3 in Appendix G of the *PostScript Language Reference Manual, Second Edition*.

| | | | |
|---|---|---|---|
| ZFold | Saddle | DoubleGate | LeftGate |
| RightGate | Letter | XFold | |

***DefaultFoldType:** *foldOption*

This denotes the default type of fold for the folding mechanism. The value must be one of the options listed under *FoldType or it will be Unknown.

***?FoldType:** "*query*"

This query returns a string denoting the current type of fold on the folding mechanism. The value returned must be one of the options listed under *FoldType or it will be Unknown.

***FoldWhen** *foldOption*: *"invocation"* UI

This keyword provides the InvocationValue to control when a job is folded, if folding has been invoked. Examples include "end of job", "end of group." One of the options must be None or False, to turn off folding.

These *foldOptions* are used with the *FoldWhen keyword to determine when the document should be folded:

- None—Do not fold.
- DeviceDeactivation—Fold immediately after the device has been deactivated.
- EndOfJob—Fold when the last page has joined the other pages in the job, so the entire job can be folded together. The notion of "job" is explained in section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfSet—Fold when the last page has joined the other pages in the set, so the entire set can be folded together. The definition of "set" depends on whether the document is collated. For a definition of "set," see the **NumCopies** and **Collate** entries in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfPage—Fold after each **showpage** or **copypage**.

**F*DefaultFoldWhen:** *foldOption*

This denotes the default for when the folding mechanism will fold a job. The value must be one of the options listed under *FoldWhen or it will be Unknown.

***?FoldWhen:** "*query*"

This query returns a string denoting the current state of when the folding mechanism will fold a job. The value returned must be one of the options listed under *FoldWhen or it will be Unknown.

***Sorter** *sortOption: "invocation"* UI

This keyword provides the InvocationValue to invoke sorting. On some devices, there might be different kinds of sorting; on other devices, sorting may simply be on or off. One of the options must be None or False to turn off sorting.

The currently registered values for *sortOption* are:

- True—Turn on sorting.
- False—Turn off sorting.

***DefaultSorter:** *sortOption*

This denotes the default state of the sorter mechanism. The value must be one of the options listed under *Sorter or it will be Unknown.

***?Sorter:** **"***query"*

This query returns a string denoting the current state of the sorter mechanism. The value returned must be one of the options listed under *Sorter or it will be Unknown.

***StapleLocation** *stapleOption*: "*invocation*" UI

This keyword provides an InvocationValue that controls where the staple is placed on the page—for devices where the location is expressed as a single parameter. A PPD file will contain either *StapleLocation or *StapleX and *StapleY but not both. *StapleLocation should be ignored by a print manager unless *StapleWhen has been invoked with a value other than None.

***DefaultStapleLocation:** *stapleOption*

This denotes the default location for stapling. The value must be one of the options listed under *StapleLocation or it will be Unknown.

***?StapleLocation:** "*query*"

This query returns a string that denotes the current stapling location. It will return one of the options listed under *StapleLocation or it will be Unknown.

The following *stapleOptions* are used with the *StapleLocation keyword to determine the location of staples:

- SinglePortrait—With the page in portrait orientation, a single staple is put at the upper left.
- SingleLandscape—With the page in landscape orientation, a single staple is put at the upper left.
- DualLandscape—With the page in landscape orientation, two staples are put along the top edge of the page, approximately 1/3 and 2/3 of the way across the page, respectively.

***StapleX** *stapleOption*: "*invocation*" UI

This keyword provides an InvocationValue that controls the *x* dimension (in default user space) of where the staple is placed on the page—for devices where the location is expressed as two parameters, *x* and *y*. This keyword must appear in PPD files in which *StapleY appears. A PPD file will contain either *StapleLocation or *StapleX and *StapleY but not both. *StapleX should be ignored by a print manager unless *StapleWhen has been invoked with a value other than None.

These *stapleOptions* are used with the *StapleX keyword to determine the location of staples in relation to the *x* axis when the page is in portrait orientation:

- Left—The staple is placed along the left side of the page. Exactly where it is placed in relation to the left edge is device-dependent.
- Right—The staple is placed along the right side of the page. Exactly where it is placed in relation to the right edge is device-dependent.
- Saddle—The staple is placed halfway along the *x* axis of the page. This is commonly used when the page is to be stapled along the center and then folded in half along the staple line to form a booklet.

***DefaultStapleX:** *stapleOption*

This denotes the default location for stapling. The value must be one of the options listed under *StapleX or it will be Unknown.

***?StapleX:** "query"

This query returns a string that denotes the current stapling location. It will return one of the options listed under *StapleX or it will be Unknown.

***StapleY** *stapleOption*: "*invocation*" UI

This keyword provides an InvocationValue that controls the *y* dimension (in default user space) of where the staple is placed on the page—for devices where the location is expressed as two parameters, *x* and *y*. This keyword must appear in PPD files in which *StapleX appears. A PPD file will contain either *StapleLocation or *StapleX and *StapleY but not both. *StapleY should be ignored by a print manager unless *StapleWhen has been invoked with a value other than None.

These *stapleOptions* are used with the *StapleY keyword to determine the location of staples in relation to the *y* axis with the page in portrait orientation:

- Top—The staple is placed at the top of the page. Exactly where it is placed in relation to the top edge is device-dependent
- OneThird—The staple is placed 1/3 of the way down the page.
- Middle—The staple is placed halfway down the page.
- TwoThirds—The staple is placed 2/3 of the way down the page.
- Bottom—The staple is placed at the bottom of the page. Exactly where it is placed in relation to the bottom edge is device-dependent

***DefaultStapleY:** *stapleOption*

This denotes the default location for stapling. The value must be one of the options listed under *StapleY or it will be Unknown.

***?StapleY:** "*query*"

This query returns a string that denotes the current stapling location. It will return one of the options listed under *StapleY or it will be Unknown.

***StapleWhen** *stapleOption*: "*invocation*" UI

This keyword provides the InvocationValue to control when a job is stapled, if stapling has been invoked. Examples include "end of job," "end of group." One of the options must be None or False to turn off stapling.

***DefaultStapleWhen:** *stapleOption*

This denotes the default time for stapling. The value must be one of the options listed under *StapleWhen or it will be Unknown.

***?StapleWhen:** "*query*"

This query returns a string that denotes when stapling will occur under the current setting. It will return one of the options listed under *StapleWhen or it will be Unknown.

These *stapleOptions* are used with the *StapleWhen keyword to determine when the document should be stapled:

- None—Do not staple.
- DeviceDeactivation—Staple immediately after the device has been deactivated.
- EndOfJob—Staple when the last page has joined the other pages in the job, so the entire job can be stapled together. The notion of "job" is explained in section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfSet—Staple when the last page has joined the other pages in the set, so the entire set can be stapled together. The definition of "set" depends on whether or not the document is collated. For a definition of "set," see the **NumCopies** and **Collate** entries in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfPage—Staple after each **showpage** or **copypage**.

***StapleOrientation** *orientationOption*: *"invocation"*

UI

This keyword provides the InvocationValue to control the orientation of the staple; for example, 45 degrees. *StapleOrientation should be ignored by a print manager unless *StapleWhen has been invoked with a value other than None.

These *stapleOptions* are used with the *StapleOrientation keyword to determine the orientation of the staple with respect to default user space:

- 0—The staple is not turned. That is, the staple is horizontal, or parallel to the *x* axis of the page.
- 45—The staple is rotated 45 degrees clockwise from the *x* axis of the page.
- 90—The staple is rotated 90 degrees clockwise from the *x* axis of the page. That is, the staple is vertical, or parallel to the y axis of the page.
- 135—The staple is rotated 135 degrees clockwise from the *x* axis of the page.

***DefaultStapleOrientation:** *orientationOption*

This denotes the default orientation for the staple. The value must be one of the options listed under *StapleOrientation or it will be Unknown.

***?StapleOrientation:** "*query*"

This query returns a string that denotes the current orientation for stapling. It will return one of the options listed under *StapleOrientation or it will be Unknown.

***BindEdge** *bindOption*: "*invocation*"

UI

This provides the InvocationValue to control which edge is bound. One of the options must be None to turn off binding. *BindEdge should be ignored by a print manager unless *BindWhen has been invoked with a value other than None.

These *bindOptions* are used with the *BindEdge keyword to determine the location of binding relative to the page in default user space (portrait orientation):

- Left—The binding is placed along the left side of the page.
- Right—The binding is placed along the right side of the page.
- Bottom—The binding is placed along the bottom of the page.
- Top—The binding is placed along the top of the page.

***DefaultBindEdge:** *bindOption*

This denotes the default edge for binding. The value must be one of the options listed under *BindEdge or it will be Unknown.

***?BindEdge:** "*query*"

This query returns a string denoting which edge will be bound under the current setting. The value returned must be one of the options listed under *BindEdge or it will be Unknown.

***BindType** *bindOption*: *"invocation"*

UI

This provides the InvocationValue to control the type of binding. *BindType should be ignored by a print manager unless *BindWhen has been invoked with a value other than None.

***DefaultBindType:** *bindOption*

This denotes the type of binding. The value must be one of the options listed under *BindType or it will be Unknown.

***?BindType:** "*query*"

This query returns a string denoting which type of binding will occur under the current setting. The value returned must be one of the options listed under *BindType or it will be Unknown.

***BindColor** *colorOption*: "*invocation*"

UI

This provides the InvocationValue to control the binding color. The valid values for *colorOption* are strings describing the color of the binding. These strings vary from product to product. *BindColor should be ignored by a print manager unless *BindWhen has been invoked with a value other than None.

***DefaultBindColor:** *colorOption*

This denotes the default color of binding. The value must be one of the options listed under *BindColor or it will be Unknown.

***?BindColor:** "*query*"

This query returns a string denoting which binding color will be used under the current setting. The value returned must be one of the options listed under *BindColor or it will be Unknown.

***BindWhen** *bindOption*: *"invocation"*

UI

This keyword provides the InvocationValue to turn on binding and to control when a job is bound. One of the options must be None or False to turn off binding.

***DefaultBindWhen:** *bindOption*

This denotes the default time for binding. The value must be one of the options listed under *BindWhen or it will be Unknown.

***?BindWhen:** *"query"*

This query returns a string that denotes when binding will occur under the current setting. It will return one of the options listed under *BindWhen or it will be Unknown.

These *bindOptions* are used with the *BindWhen keyword to determine when the document should be bound:

- None—Do not bind.
- DeviceDeactivation—Bind immediately after the device has been deactivated.
- EndOfJob—Bind when the last page has joined the other pages in the job, so the entire job can be bound together. The notion of "job" is explained in section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfSet—Bind when the last page has joined the other pages in the set, so the whole set can be bound together. The definition of "set" depends on whether or not the document is collated. For a definition of "set," see the **NumCopies** and **Collate** entries in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.

***Booklet** *bookletOption*: *"invocation"* UI

This provides the InvocationValue to make booklets. Booklets are created by saddle stitching, folding, and trimming. One of the options must be None or False, to turn off booklet-making.

The currently registered values for *bookletOption* are:

- True—Make a booklet.
- False—Do not make a booklet.

**F*DefaultBooklet:** *bookletOption*

This denotes the default state of booklet making. The value must be one of the options listed under *Booklet or it will be Unknown.

***?Booklet:** "*query*"

This query returns a string denoting the current state of booklet making. The value returned must be one of the options listed under *Booklet or it will be Unknown.

***Slipsheet** *slipsheetOption*: "*invocation*"

UI

This provides the InvocationValue to control slipsheeting. Slipsheeting is the insertion of pages of a different color or type between sets of documents. One of the options must be None or False to turn off slipsheeting.

The currently registered values for *slipsheetOption* are:

- None—Turn off slipsheeting.
- DeviceDeactivation—Insert slipsheet at device deactivation.
- EndOfJob—Insert slipsheet at the end of the current job.
- EndOfSet—Insert slipsheet at the end of the current set.
- True—Turn on slipsheeting—for devices in which slipsheeting is a binary state. Whether this activates slipsheeting at the end of the job, end of set, or device deactivation is device-dependent.
- False—Turn off slipsheeting—for devices in which slipsheeting is a binary state.

***DefaultSlipsheet:** *slipsheetOption*

This denotes the default state of slipsheeting. The value must be one of the options listed under *Slipsheet or it will be Unknown.

***?Slipsheet:** "*query*"

This query returns a string denoting the current state of slipsheeting. The value returned must be one of the options listed under *Slipsheet or it will be Unknown.

**F*InsertSheet** True | False: *"invocation"*

UI

This provides the InvocationValue to insert a sheet at a specific place in the document. For example, a printer might allow the insertion of a photograph between specific pages of the document after the pages have passed through the heated elements in the printer. To accomplish this, a print manager would emit the code for a True value at the beginning of the specific page, emit the **showpage** operator to insert the special sheet, and then emit the code for the False value of *InsertSheet.

***DefaultInsertSheet:** True | False | Unknown

This denotes the default state of *InsertSheet. A value of True means that the next page will be drawn from a special input tray and inserted in the page sequence. A value of False means that the next page will be drawn from one of the regular input trays.

***?InsertSheet:** *"query"*

This query returns a string denoting the current state of *InsertSheet. The value returned must be one of the options listed under *InsertSheet or it will be Unknown.

***Jog** *jogOption:"invocation"* UI

This provides the InvocationValue to control jogging.When jogging is invoked, the next job or set is offset to the left or right from the previous job or set in the output bin. Jogging is also known as "offset stacking". One of the options must be None or False to turn off jogging.

The currently registered values for *jogOption* are:

- None—Turn off jogging.
- DeviceDeactivation— Jog at device deactivation.
- EndOfJob— Jog at the end of the current job.
- EndOfSet— Jog at the end of the current set.

***DefaultJog:** *jogOption*

This denotes the default state of jogging. The value must be one of the options listed under *Jog or it will be Unknown.

***?Jog:** *"query"*

This query returns a string denoting the current state of jogging. The value returned must be one of the options listed under *Jog or it will be Unknown.

## 5.15 Imagesetter Features

This section contains features that are usually found only on imagesetters (also referred to as typesetters and filmsetters). These features are implemented by device-dependent means, but a uniform interface to them is provided by the PostScript interpreter. Each of these features is documented in section 4.11 of the *PostScript Language Reference Manual, Second Edition*.

***MirrorPrint** True | False: *"invocation"* UI

This keyword provides the InvocationValue to turn the mirror print feature on (True) and off (False).

***DefaultMirrorPrint:** True | False

This denotes the default state of mirror printing.

***?MirrorPrint:** *"query"*

This query returns True if the device is currently set up to print mirror prints; False if it is not.

***NegativePrint** True | False: *"invocation"* UI

This keyword provides the InvocationValue to turn the negative print feature on (True) and off (False).

***DefaultNegativePrint:** True | False

This denotes the default state of negative printing.

***?NegativePrint:** *"query"*

This query returns True if the device is currently set up to print negative prints; False if it is not.

***AdvanceMedia** *advanceOption*: *"invocation"*

UI

This keyword provides the InvocationValue to tell the device when to advance roll-fed media by a preset distance.

The currently registered values for *advanceOption* are:

- None—Do not advance the medium.
- DeviceDeactivation—Advance the medium at device deactivation.
- EndOfJob—Advance the medium at the end of the job.
- EndOfSet—Advance the medium after each set.
- EndOfPage—Advance the medium after each **showpage** or **copypage.**

***DefaultAdvanceMedia:** *advanceOption*

This denotes the default state of *AdvanceMedia.The value must be one of the options listed under *AdvanceMedia or it will be Unknown.

***?AdvanceMedia:** *"query"*

This query returns a string denoting the current state of *AdvanceMedia. The value returned must be one of the options listed under *AdvanceMedia or it will be Unknown.

***CutMedia** *cutOption*: *"invocation"*

UI

This keyword provides the InvocationValue to tell the device when to cut roll-fed media.

The currently registered values for *cutOption* are:

- None—Do not cut the medium.
- DeviceDeactivation—Cut the medium at device deactivation.
- EndOfJob—Cut the medium at the end of the job.
- EndOfSet—Cut the medium after each set.
- EndOfPage—Cut the medium after each **showpage** or **copypage.**

***DefaultCutMedia:** *cutOption*

This denotes the default state of *CutMedia.The value must be one of the options listed under *CutMedia or it will be Unknown.

***?CutMedia:** *"query"*

This query returns a string denoting the current state of *CutMedia. The value returned must be one of the options listed under *CutMedia or it will be Unknown.

## 5.16 Resolution and Appearance Control

This section contains keywords that control the resolution and related appearance characteristics of the device.

***DefaultResolution:** *resolutionOption* | Unknown

This entry provides the default resolution of the device, in dots (spots) per linear inch, in both *x* and *y* dimensions and in PostScript default user space. The value *resolutionOption* must be a string either of the form 300dpi or of the form 300x300dpi, or it can be Unknown if the resolution cannot be determined upon power-up. The value of *resolutionOption* appearing here must be a valid resolution listed under *SetResolution or *Resolution.

If the format of *resolutionOption* is 300x300dpi, the device supports anamorphic resolution; that is, the resolution in the *x* dimension can be different from the resolution in the *y* dimension. For example, a printer might support a resolution of 300x600dpi. The first number denotes the resolution in the *x* dimension; the second number denotes the resolution in the *y* dimension. The "x" in the middle is a convenient separator, and the dpi signifies "dots per inch."

The format 300dpi is a shorthand form of 300x300dpi and means that the resolution is the same in both the x and y dimensions (the device does not support anamorphic resolution). This is the most common case found in PPD files.

The format of *resolutionOption* used by *DefaultResolution must be used consistently wherever a *resolutionOption* appears. The two formats 300dpi and 300x300dpi cannot be intermixed in a PPD file.

***Resolution** *resolutionOption*: *"invocation"* UI

For devices that support resolution changes from within a PostScript language job, this entry will provide the proper InvocationValue for each resolution supported by the device. There can be several of these entries, if the

PostScript device supports multiple selectable resolutions. The string *resolutionOption* is of the form specified in the *DefaultResolution entry. Print managers need to ensure that any resolution changes occur before the page size is selected.

*Note* **Resolution does not require a password to precede the invocation. If a device requires a password to change the resolution, the PPD file should contain *SetResolution, instead of *Resolution.*

† ***SetResolution** *resolutionOption*: *"invocation"*

For devices that support resolution changes from software and require that the resolution be changed "outside the server loop," in initial virtual memory, this entry will provide the proper InvocationValue for each resolution supported by the device. There can be several of these entries, if the PostScript device supports multiple selectable resolutions. The string *resolutionOption* is of the form specified under *DefaultResolution. Print managers need to ensure that any resolution changes occur before the page size is selected.

*Note* **SetResolution requires a password to precede the invocation, and thus should be present only in the PPD files of devices that require a password to change the resolution. Devices that do not require a password to change the resolution should use *Resolution.*

***?Resolution:** *"query"*

This query returns a string denoting the current resolution of the device. The returned value will be a string of the form specified under *DefaultResolution or it will be Unknown. The resolution returned must be a valid resolution listed under *SetResolution or *Resolution.

† This keyword requires the *Password value to be supplied in front of the invocation.

***Smoothing** *smoothOption*: *"invocation"*

UI

This provides the InvocationValues to invoke various levels of "smoothing" the edges of text and graphics after they have been rendered by the device. Smoothing is also sometimes referred to as "bit smoothing," "anti-aliasing," or "resolution enhancement," Option keywords describe the level of smoothing. One of the options must be None or False to turn off smoothing.

The currently registered values for *smoothOption* are:

- None—No smoothing.
- Light—Turn on light smoothing.
- Medium—Turn on medium smoothing
- Dark—Turn on dark smoothing
- True—Turn on smoothing (for a device that has only a binary setting).
- False—Turn off smoothing (for a device that has only a binary setting).

***DefaultSmoothing:** *smoothOption*

This denotes the default state of the smoothing mechanism. The value must be one of the options listed under *Smoothing or it will be Unknown.

***?Smoothing:** *"query"*

This query returns a string that denotes the current state of the smoothing mechanism. The returned value must be an option listed under *Smoothing or it may be Unknown.

***BitsPerPixel** *depthOption*: *"invocation"*

UI

This provides the InvocationValues to select various gray-scale levels or color depths. Option keywords are strings that denote the number of bits per pixel that should be used to represent a color when rendering the job on the device. One of the options must be None, which is equivalent to 1 bit per pixel.

The currently registered values for *depthOption* are:

- None—Use 1 bit per pixel.
- 2—Use 2 bits per pixel.
- 4—Use 4 bits per pixel.
- 8—Use 8 bits per pixel.

***DefaultBitsPerPixel:** *depthOption*

This denotes the default number of bits per pixel used to render color or gray-scale on this device. The value must be one of the options listed under *BitsPerPixel or it will be Unknown.

***?BitsPerPixel:** *"query"*

This query returns a string that denotes the current number of bits per pixel used to render color or gray-scale on this device. The returned value must be an option listed under *BitsPerPixel or it will be Unknown.

## 5.17 Gray Levels and Halftoning

***AccurateScreensSupport:** True | False

This string value indicates whether or not the device supports Adobe's Accurate Screens technology. The value is True if accurate screens are supported, otherwise it is False. The accurate screens feature is documented in section 6.4 of the *PostScript Language Reference Manual, Second Edition*.

***ScreenFreq:** *"real"*

This QuotedValue is the *frequency* argument returned by the **currentscreen** operator after powering on the device. It represents the halftone screen frequency.

***ScreenAngle:** *"real"*

This QuotedValue is the *angle* argument returned by the **currentscreen** operator after powering on the device. It represents the halftone screen angle.

***ScreenProc** *spotOption*: *"{ procedure }"*

This InvocationValue provides a procedure body that is suitable for use as a "spot function" with the **setscreen** or **sethalftone** (*Level 2*) operator. The *spotOption* represents the name of the spot function. These options are used to specify an alternate shape for the halftone spot. There can be one or more of these spot shape options in a PPD file.

***DefaultScreenProc:** *spotOption*

This represents the proc argument (the default halftone spot function) returned by the **currentscreen** operator after powering on the device. The string *spotOption* must correspond to one of the options listed under *ScreenProc.

These spot options are used by the *ScreenProc keyword to set the halftone screen spot function. Any of these options can also have a .Inverse qualifier, which would invert the color of the spot function, or it can have a serialization qualifier to distinguish it from other options.

The currently registered values for *spotOption* are:

- Dot—This keyword represents a standard dot-shaped halftone screen function. This is the default shape for the halftone cell on many PostScript language implementations, and basically consists of small, black, roughly circular spots that vary in size with the gray level. This keyword also encompasses more sophisticated functions that also produce circular dots (for example, as found on higher-resolution devices), but which might slightly differ from the most basic dot screen.

- Line—This keyword represents a line screen halftone function. Gray levels will be rendered by parallel lines which vary in thickness according to the gray level.

- Ellipse—This keyword provides an “elliptical spot” screen, which is similar to a dot screen except that the dots are elliptical rather than circular.

- Cross—This provides a “crosshatch” screen halftone function.

- Mezzo—This provides a pseudorandom “mezzotint” screen function for the halftone mechanism.

- DiamondDot—This provides a screen in which low gray levels produce round dots, medium gray levels produce diamond-shaped dots, and high gray levels produce negative dots. This screen produces smoother transitions among medium gray levels.

***Transfer** *transferOption*: *“{ procedure }”*

This keyword provides InvocationValues for possible transfer functions which may be invoked with the operators **settransfer**, **setcolortransfer**, and **sethalftone** (Level 2 only). A transfer function is a procedure that corrects for the characteristics of a particular marking engine or display technology to obtain “true” optical gray or color densities. A transfer function is expected to return accurate results at the 10% increments and should return reasonable values at any point between 0 and 1.

Since transfer functions are inherently specific to an *instance* of a type of device, any transfer functions should be entered into a local customization file for a specific device. Most PPD files will ship without any transfer functions defined for a class of devices.

*Note* *When transfer functions are used at the PostScript language level, always concatenate the transfer function with the existing one, rather than replacing it. See section 6.3 of the PostScript Language Reference Manual, Second Edition for more information about transfer functions and their uses.*

***DefaultTransfer:** Null | Factory

On monochrome devices, this is the built-in transfer function, as returned by the **currenttransfer** operator immediately after powering up the device. Most devices ship with a null default transfer function.

Any of these transfer option keywords can also have the .Inverse qualifier or a serialization qualifier to distinguish it from other options. Inversion is typically performed by appending **1 exch sub** to the existing transfer function, but an inverse normalized function can be much more complex.

The currently registered values for *transferOption* are:

- Null—This is provided to indicate a null procedure body for the transfer function. A null procedure body is represented in the PostScript language as a pair of curly braces, **{ }**.
- Factory—For a monochrome device that ships from the factory with a built-in non-null transfer function, this entry lists the transfer function built into the device.
- Normalized—For a monochrome device, this provides a normalized transfer function to obtain “true” optical gray densities. For a color device, the Normalized option provides the transfer function to correct the gray values on an RGB device and the black colorant on a CMYK device.
- Red—For a color device, this provides a normalized transfer function to correct the red colorant on an RGB device or the cyan colorant on a CMYK device.
- Green—For a color device, this provides a normalized transfer function to correct the green colorant on an RGB device or the magenta colorant on a CMYK device.
- Blue—For a color device, this provides a normalized transfer function to correct the blue colorant on an RGB device or the yellow colorant on a CMYK device.

## 5.18 Color Issues

This section contains keywords used to adjust colors on color devices.

***BlackSubstitution** True | False: *"invocation code"* UI

This provides the InvocationValue to invoke black substitution. When True, it indicates that the printer should substitute process black ink for any pixel that is marked in composite black (cyan, magenta, and yellow inks all requested) to produce better black.

***DefaultBlackSubstitution:** True | False

This denotes the default state of the black substitution feature.

***?BlackSubstitution:** *"query"*

This query returns True if black substitution is currently invoked and False if it is not.

***ColorModel** *colormodelOption*:*"invocation"* UI

This keyword provides InvocationValues to select different native color models to be used by the device for imaging. The native color model is the color model to which all colors are converted before rendering.

***DefaultColorModel:** *colormodelOption*

This denotes the default native color model of the device. The value must be one of the options listed under *ColorModel or it will be Unknown.

***?ColorModel:** *"query"*

This query returns the current native color model. The returned value must match one of the options under *ColorModel or it will be Unknown.

The currently registered values for *colormodelOption* are

- CMY—Cyan-magenta-yellow color model.
- CMYK—Cyan-magenta-yellow-black color model.
- RGB—Red-green-blue color model.
- Gray—Gray-scale color model.

***ColorRenderDict** *dictOption*: *"invocation"*

On Level 2 color devices, manufacturers can supply different built-in color rendering dictionaries to calibrate the device colors for different types of paper, different halftone screens, or for many other purposes. The keyword *ColorRenderDict provides the InvocationValue to set up the new color rendering dictionary referred to by *dictOption*.

The option *dictOption* will typically be an integer because there is no logical name for each dictionary, so it will also typically have a translation string representing a more meaningful dictionary name to display to the user.

For example:

```
*ColorRenderDict 1/Bond Paper: "
  /CRD1 /ColorRendering findresource
  setcolorrendering"
*End
```

An instance of this keyword will be supplied for each built-in color rendering dictionary. The code for this keyword will invoke the named color rendering dictionary.

In addition, if a user wants to supply new color rendering dictionaries, new instances of this keyword can be added to a local customization file for a given device. For new dictionaries in the local customization file, the code would have to create the dictionary and fill it with the appropriate values for color calibration. For example:

```
*ColorRenderDict 2/Paper Towels: "10 dict dup begin
...
setcolorrendering"
*End
```

The "..." represents key-value pairs that are created and put into the dictionary by this code. The key-value pairs represent color calibration values.

## 5.19 Color Separation Keywords

Color separations are device-dependent. A color separation is a monochrome print that represents a single color plate that is later printed in combination with other plates on a full color press system. In this sense, a color separation can be one of the four standard *process colors* (cyan, magenta, yellow, and black) from which all other colors are simulated by mixing, or it can be a particular *spot color,* which is simply an ink of a particular color.

For color separations to work well, it must be possible to print several layers one on top of the other on a color printing press. The way the color mixing is optimized is to print each color plate with a different halftone screen, usually rotated at some specific angle to minimize both dot interference with other plates and to avoid moiré patterns.

The selection of these halftone screens is typically done by hand for a particular device, taking resolution and other device characteristics into account (even variations in the speed of media travel). Once a good set of screen parameters have been established, they are used for almost all separations on that machine, unless screens of different granularity are desired, in which case the process is repeated.

In addition to the halftoning process necessary for producing separations, there are issues of color matching that are equally device-dependent. As an example, many companies have specific names for their entire range of colored inks. These colors can be simulated or approximated with various color technologies (screen phosphors or process inks) but it might not be possible to render them exactly. There is usually a color mapping table that associates a particular combination of process inks (or screen phosphor intensities) to one of the named colors. This is, of course, device-specific.

### Option Keywords

Color separation option keywords (the notation *colorsepkey* in the keyword listings) are designed to reflect particular combinations of separation characteristics. For example, a given separation typically is designed for a particular process color (for example, the cyan separation), at a certain halftone screen frequency, for a particular resolution device. To this end, the color separation option keywords are complex and modular, but they can be made more human-readable through use of the general translation string mechanism provided in the file format.

A *colorsepkey* consists of a name that can optionally have any number of qualifiers (sub-components), each separated by a dot (., decimal ASCII 46). The key is typically a color name, and the qualifiers typically refer to a screen frequency, a resolution, and sometimes to vendor-specific or printer-specific features that can affect the appearance of the color separation, such as a special screening method or a specific type of controller.

Two common qualifiers are defined in this spec: screen frequency, which must end in the string lpi, and resolution, which must end in the string dpi. These qualifiers occur in the following relationship:

```
colorname.frequency.resolution
```

Any number of other qualifiers can appear after the resolution qualifier and will be separated from each other by a dot.

The idea is to be able to associate many different components of a color separation package by keyword. The keywords are arbitrary, but the structured qualifiers make it possible for an application to separate the components, if necessary, to allow a user to choose from several frequencies, optional resolutions, and so on. Otherwise, these keywords behave similarly to any other option keywords in PPD files. For devices where the resolution cannot be varied (most of them), the resolution qualifier will usually be omitted from the *colorsepkey* keyword.

The following are several examples, to help illustrate the format more clearly:

```
*ColorSepScreenAngle ProcessCyan.60lpi.1270dpi: "37"
*ColorSepScreenAngle ProcessMagenta.60lpi.1270dpi: "45"
*ColorSepScreenAngle ProcessYellow.60lpi.1270dpi: "75"
*ColorSepScreenAngle ProcessBlack.60lpi.1270dpi: "0"
*ColorSepScreenFreq ProcessBlack.60lpi.1270dpi: "60"
*ColorSepScreenProc ProcessBlack.60lpi.1270dpi: "{ pop }"
*ColorSepTransfer ProcessBlack.60lpi.1270dpi: "{ 1 exch sub }"
*ColorSepScreenFreq ProcessCyan.90lpi.1270dpi: "90"
*ColorSepScreenFreq ProcessCyan.60lpi.600dpi: "60"
```

The following keywords provide suggested values for manipulating the PostScript language halftone machinery to provide good color separations. Each separate process color should be printed with a different screen angle and perhaps different transfer functions or at various screen frequencies.

Be aware that all color separation entries are optional. If an entry does not exist for a specific color, the default value should be used. For example, there might be entries for screen frequencies and screen angles for a color but not an entry for a screen procedure for that color.

***DefaultColorSep:** *colorsepkey*

This keyword provides the default color separation in the form of a *colorsepkey* keyword. This is used in conjunction with the other entries listed below.

***ColorSepScreenFreq** *colorsepkey: "real"*

This keyword provides the InvocationValue for the appropriate screen frequency for a color separation keyed to the given *colorsepkey*.

***ColorSepScreenAngle** *colorsepkey: "real"*

This entry gives the halftone screen angle InvocationValue for the given color separation.

***ColorSepScreenProc** *colorsepkey: "{procedure}"*

This provides the halftone spot function InvocationValue for the specified color separation.

***ColorSepTransfer** *colorsepkey: "{procedure}"*

This entry provides the transfer function InvocationValue appropriate for the given color separation keyword.

***CustomCMYK** *inkname: "cyan magenta yellow black"*

This entry provides an InvocationValue containing the CMYK equivalents for a named custom color. These can be user-defined or names used in a commercial color matching system that can provide CMYK approximations for particular marking technologies. The idea is to associate any given named ink (whether it be from a commercial color matching system or a local custom color) with a set of process color values to approximate it. For example:

```
*CustomCMYK HarvestGold: "0 .01 .9 .01"
```

The keyword *CustomCMYK is kept deliberately brief because there might be hundreds of entries of this sort in a PPD file. For some devices, in fact, these entries can be put into a separate file (considered a customization of the standard file) and reference the original with the *Include convention mentioned in section 2.6.

***InkName:** *inkname* / *alias*

This keyword provides a StringValue that is an alternative name for one of the inkname keywords used in the *CustomCMYK section. It provides slightly more human-readable versions of the keywords that can be presented in a user interface (the keywords themselves cannot contain spaces). Here is an example:

```
*InkName: p305/COLORNAME 305
```

Alternatively, you can omit the *InkName entry and simply supply a translation string for the option keyword of the *CustomCMYK entry. For example:

```
*CustomCMYK p305/Harvest Gold 305: "0 .01 .9 0.1"
```

***Separations** True | False: *"invocation"* UI

If the device can provide automatic generation of color separations, this provides the InvocationValue to tell the device to output either color separations (True) or composite color (False). True means that this device will produce each page by printing multiple color separations, one for each device colorant. False means that the device will produce each page as a single composite page with all the colors, if any, combined on the same page. Color separations are explained in section 4.8 of the *PostScript Language Reference Manual, Second Edition*.

***DefaultSeparations:** True | False

This denotes the default state of the automatic separations mechanism.

***?Separations:** *"query"*

This query will return True if the device is set up to produce color separations of a given file, and False if the device is set up to produce all colors on a single page.

## 5.20 Font Related Keywords

This section contains keywords that provide information about the fonts on the device.

***Font** *fontname*: *encoding "(version)" charset status*

This entry provides a StringValue that gives one line of information for each font resident on the product. An initial PPD file will contain only the fonts shipped with the product in its minimal configuration. These fonts may be in ROM or on a peripheral device such as a hard disk.

The option *fontname* is the valid PostScript language name of the font, without the leading slash. The value of *encoding* is a keyword indicating the type of encoding. A list of valid *encoding* values can be found at the end of this section.

The *version* field contains the version number of the font (as found under the key *version* in the **FontInfo** dictionary that is a subdictionary of the particular font dictionary). The *charset* field indicates the character set of the font, which may be different from the encoding of the font. A list of valid *charset* options can be found at the end of this section.

The *status* field indicates whether or not the font can be removed without causing the printer to cease its normal functioning.Valid keywords for the *status* field are ROM and Disk. The distinction between ROM and Disk is that upon powering up the device, a font from the ROM list will be inaccessible only if there is a printer malfunction. A font from the Disk list, while usually available, could possibly be inaccessible without a printer malfunction.

The following table contains examples of typical cases of font distribution and under which *status* keyword they would be assigned. Note that this table is not exhaustive as to the different methods of font distribution; more exist than are documented here.

**Table 7** *Designation of fonts: ROM versus Disk*

| *Font distribution* | *Erasable* | *Removable* | *Status* |
|---|---|---|---|
| ROM-resident | No | No | ROM |
| auto-loaded into RAM, read-only | No | No | ROM |
| internal read-only CD-ROM | No | No | ROM |
| downloaded to RAM, writable | Yes | No | Disk |
| external read-only CD-ROM | No | Yes | Disk |
| internal writable hard disk | Yes | No | Disk |
| external writable hard disk | Yes | Yes | Disk |

It is possible for a device to ship with all fonts listed under Disk. For example, a device can exist on which all fonts are shipped on an external CD-ROM (the fonts are not erasable, but the CD-ROM is removable, therefore these fonts would be listed under Disk).

Fonts that are added to the printer any time after its initial shipment from the factory can be added to a local customization file for the printer. Such fonts will usually have a *status* field of Disk, as it is highly likely that aftermarket fonts would be freely removable.

*Note* *The charset and status fields of the *Font keyword are new in version 4.0 of this specification. They did not exist in version 3.0.*

### Font Encoding Options

The *encoding* value of the *Font keyword indicates the default encoding of each font. Fonts can be re-encoded to provide other encodings; the character set options for each font indicate which encodings are possible for that font. The following are the current *encoding* options:

- Standard—This indicates a font which, by default, uses the Adobe **StandardEncoding** vector.
- Special—This indicates a font with nonstandard font-specific encoding (for instance, the Sonata font).
- ISOLatin1—This indicates a font which, by default, uses the Adobe **ISOLatin1Encoding** vector.
- Expert—This indicates a font which, by default, uses the Adobe Expert encoding vector.

- ExpertSubset—This indicates a font which, by default, uses the Adobe ExpertSubset encoding vector.

- JIS—This indicates a Japanese font with JIS (Japan Industrial Standard) encoding. (This is a two byte-per-character encoding.)

- RKSJ—This indicates a Japanese font with RKSJ (Romaji-Kana-Shift-JIS) encoding. (This is a mixed one and two byte-per-character encoding, common on PCs, and often informally referred to as “Shift JIS.” In this specification, “Shift-JIS” refers to the two byte-per-character encoding, which is a proper subset of RKSJ.)

- EUC—This indicates a Japanese font with EUC (Extended UNIX Code) encoding. (This is a two byte-per-character encoding.)

- Shift-JIS—This indicates a Japanese font with Shift-JIS encoding. (This is a two byte-per-character encoding. It is a proper subset of RKSJ. The Japanese PC encoding commonly referred to as “Shift JIS,” which includes one-byte Romaji and Katakana codes, is referred to in this specification as RKSJ.)

### Character Set Options

The *charset* value of the *Font keyword indicates which shape descriptions (glyphs) are contained in the font and are available for re-encoding. For example, most Roman fonts from Adobe contain enough glyphs to support both the Standard and ISOLatin1 encodings, and so would be labeled Standard.

The distinction between encodings and character sets can be confusing. For most Roman fonts, a given encoding corresponds to one, or at most two, character sets. For example, Roman fonts that use Standard encoding will have either the Standard or OldStandard character set. For Japanese fonts, on the other hand, almost all combinations of encoding and character set are supported.

- Standard—This indicates a Roman font that contains the character set that supports both the Standard and ISOLatin1 encodings.

- OldStandard—This indicates a Roman font that contains the character set necessary to support the Standard encoding. OldStandard is a subset of the Standard character set.

- Special—This indicates a font that supports a font-specific character set (for example, Sonata).

- ISOLatin1—This indicates a Roman font that contains the character set that supports the ISOLatin1 encoding. ISOLatin1 is a subset of the Standard character set.
- Expert—This indicates a Roman font that contains the character set that supports the Expert encoding.
- ExpertSubset—This indicates a Roman font that contains the character set that supports the ExpertSubset encoding.
- JIS-83—This indicates a Japanese font that supports the JIS X0208-1983 character set.
- JIS-78—This indicates a Japanese font that supports the JIS 1978 character set.
- 83pv—This indicates a Japanese font that supports the 83pv (Apple® Macintosh-compatible) character set.
- Add—This indicates a Japanese font that supports the Add (Fujitsu FM system-compatible) character set.
- Ext—This indicates a Japanese font that supports the Extended (NEC PC-98-compatible) character set.
- NWP—This indicates a Japanese font that supports the NWP (NEC Word Processor) character set.

***DefaultFont:** Error | *fontname*

This gives the name of the default font provided by **findfont** if the requested font is not available. Note that in some devices this might not be well-defined (especially where there might be a network font server, for instance), and in these cases, this entry might not be present. For many devices this field will contain the name Courier. If this value is Error, an execution error will occur if the font is not found. Any other value implies that a font substitution will take place (such as substituting Courier).

***?FontList:** *"query"*

Provides a PostScript language sequence to return a list of all available fonts. It should consult the **FontDirectory** dictionary as well as any mass storage devices available to the device. The list does not need to be in any particular order, but each name is returned separated by a slash '/' character. This is normally the way the PostScript language **==** operator will return a font name. All white space characters should be ignored. The end of the font list is indicated by a trailing ***** sign on a line by itself (decimal ASCII 42).

The following is a look at two valid returns from the query:

```
/Optima/Optima-Bold/Optima-Oblique/Optima-BoldOblique/Courier/Symbol
*

/Courier
/Symbol
/Times-Roman
*
```

*Note* *In previous versions of this specification, using* ***flush*** *to separate names into packets was recommended. This turns out to result in major performance degradation, and is no longer recommended.*

***?FontQuery:** *"query"*

This provides a PostScript language query that should be combined with a particular list of font names being sought. It looks for any number of names on the stack, and will print a list of values depending on whether or not the font is known to the PostScript interpreter. The font names must be provided on the operand stack by the print manager. This is done by emitting the names, with leading slash '**/**' characters, before emitting the query itself. To avoid stack overflow, the number of names on the stack should be less than 150.

So that the Document Manager does not have to keep track of the precise order in which the values are returned and to guard against errors from dropped information, the syntax of the returned value will be */fontname*:Yes or */fontname*:No, where each font in the list is returned in this manner. The slashes delimit the individually returned font names, although *newlines* should be expected (and ignored) between them. A final ***** character will follow the returned values.

```
/Times-Roman:Yes
/Optima:Yes
/CircleFont:No
/Adobe-Garamond:No
*
```

The query provided by *?FontQuery is in many cases preferable to the *?FontList query, since that query can return a very long list of fonts in some devices (with mass storage such as built-in hard disks or network font servers) available to the PostScript interpreter.

## 5.21 Printer Messages

In an environment which the output device is connected to the host by a bidirectional channel, such as serial communication, the output device can return various status messages to the host. A print manager can recognize these messages and convert some of them to a more readable form before displaying them to the user. The messages are divided into categories and enumerated in the PPD file for recognition purposes.

***PrinterError:** *"string"*

Printer errors are reported automatically by the output device when something is wrong. The same printer errors can often be returned in a status message as a response to a request for status (see *Status). This provides a list of QuotedValues that are possible Printer Error messages returned by the device in the following form:

```
%%[PrinterError: cover open]%%
%%[PrinterError: paper exit misfeed]%%
```

The PPD file entries for these error messages would be as follows:

```
*PrinterError: "cover open"
*PrinterError: "paper exit misfeed"
```

The brackets, percent signs, and the word "PrinterError" from the original error message are not included in the PPD file.

If a translation string were included, the PPD file entry would look like this:

```
*PrinterError: "cover open"/lucka <F6>ppen
```

The translation string translates the error message into Swedish; the hexadecimal substring 'F6' represents the 8-bit character 'Odieresissmall'. See section 3.7 for details on translation string syntax.

***Status:** *"string"*

This lists the possible responses to a status query as QuotedValues. A status query is typically accomplished by sending ^T (control-T, decimal ASCII 20) over a serial connection or by a special status packet if a network protocol is used (for instance, AppleTalk®).

The status message may be composed of up to three parts. There is always at least the word "status: *message*" with an appropriate status message (those messages are listed in this section of the PPD file). There may also be two

other sections in the message from the device, listing the currently executing job name (job: *name*) as defined by the variable **jobname** in **statusdict**, and a source field, like this: source: *connection*.

The following are examples of status messages returned by a PostScript output device:

```
%%[status: warming up]%%
%%[status: busy; source: AppleTalk]%%
%%[job: userjob; status: waiting; source: serial25]%%
%%[job: myjob; status: PrinterError: cover open; source: serial25]%%
```

The entries in the PPD file will not have the brackets, the percent signs, or the extraneous fields for *jobname* and *source*. The PPD file will contain only the *message* field:

```
*Status: "warming up"
*Status: "busy"
*Status: "waiting"
*Status: "PrinterError: cover open"
```

Note that the message portion of a status message can contain a printer error, so the same list of printer errors that appears under *PrinterError may appear under *Status.

***Source:** *"sourceOption"*

This lists the possible sources for print jobs, as QuotedValues. These correspond to the source: field in the status message (as shown under the *Status section). The following are example entries for Level 1 devices:

```
*Source: "serial25"
*Source: "serial9"
*Source: "AppleTalk"
*Source: "Centronics"
```

and for Level 2 devices:

```
*Source: "Serial"
*Source: "SerialB"
*Source: "LocalTalk"
*Source: "Parallel"
```

The status message in which the source is found can contain other fields (as in the example under *Status), depending on the values of **jobname** in **statusdict** and whether or not there is an active job (in which case the source is listed). Just the strings for the source field are provided in this section.

***Message:** *"string"*

This provides, as QuotedValues, a list of possible device messages that do not fit into the categories of *Status, *PrinterError, or *Source. Messages that are listed under those keywords are not repeated here. The strings listed under the keyword *Message will contain the text delimiters (brackets and percent signs), if they exist in the original error message generated by the device.

The following are two examples. The first example contains the delimiters as the device generated them. The second example contains no delimiters because the device generates this message without delimiters. The second example also contains a translation string and some special syntax, which is explained below.

```
*Message: "%%[exitserver: permanent state may be changed]%%"
*Message: "\fontname\ not found, using Courier"/no \fontname\ on this printer
```

Notice the *fontname*\ notation in the last example, with the backslashes. The exact text of this message depends on which font was requested by the user program. This backslash notation is a PPD file syntax that indicates that any arbitrary PostScript language name may be found at the beginning of that message (substituted for *fontname*\). A parser, parsing a PPD file, should parse for the complete string *fontname*\. Special significance should not be given to the single character \, because a backslash can occur in other contexts.

## 5.22 System Management

† ***PatchFile:** *"invocation"*

This represents a (perhaps large) PostScript language sequence that is a downloadable patch to ROM code, which is downloaded outside the server loop, into initial VM. It is represented as a QuotedValue. It can be used if there are any known bugs in existing PostScript devices or to provide some initial state to all jobs. A program that is managing a PostScript device should make every attempt to guarantee that this information is resident in the PostScript interpreter's memory before any jobs are run.

Code in a patch file must adhere to the following requirements. These restrictions are intended to ensure that this patch code will only execute on the printer for which it was intended, and will not execute if it has already been executed on this particular printer (to conserve memory space and avoid possible conflicts). A conforming patch file must do the following:

- Check a unique key to see if the patch has already been downloaded to the printer.

† This keyword requires the *Password value to be supplied in front of the invocation.

- Compare the **product**, **version**, and **revision** strings on the printer to the values of the *Product and *PSVersion entries in the PPD file to make sure that this patch will be downloaded only to the printer for which it was written.

- If downloading the patch, define or set a unique key in a dictionary or otherwise indicate the patch's presence, so that its existence can be checked later.

***?PatchFile:** "*query*"

This query checks the key set by the code in *PatchFile and returns True if the patch file is present behind the server loop, and False if it is not. This allows a print manager to decide whether it is necessary to download the patch file outside the server loop as a separate job. The patch file's presence is determined by the presence or absence of a certain key in a dictionary, or by any other method that the implementor of the patch file chooses. If a patch file is implemented, a patch file query must be provided.

***JobPatchFile** int: "invocation"

Like *PatchFile, this is used to download a PostScript language sequence, but it does not require a password and is not downloaded outside the server loop. It is used to provide initial state for certain jobs, and should be attached to the beginning of the job and the pair of files should be downloaded as one job. The option keyword is provided so that multiple patch files on a device may be numbered.

***FreeVM:** *"int"*

This QuotedValue is the value as returned by the PostScript language sequence

```
vmstatus exch sub == pop
```

when a device is first powered on. It can be used by a document manager to determine which of several devices has more memory built into it. Free VM is a static value which does not reflect the current amount of VM available on the device. It should be regarded as a maximum limit of free VM rather than as a measure of current availability.

† This keyword requires the *Password value to be supplied in front of the invocation.

† ***Reset:** *"invocation"*

This QuotedValue is a PostScript language sequence that will perform a "soft" restart of the PostScript interpreter. It can be used by a printing manager to reboot the device under some circumstances.

***Password:** *"invocation"*

This QuotedValue provides the password required to persistently set values in initial VM on the device. It is used in conjunction with the *ExitServer keyword and other keywords that are flagged with the dagger. See section 2.6 for details on local customization for instructions on changing this password for a specific device.

† ***ExitServer:** *"invocation"*

This QuotedValue provides the appropriate PostScript language sequence to exit the job server loop (on a Level 1 device, this code would typically use the exitserver operator, and on a Level 2 device, the **startjob** operator). This should be used carefully, if at all, by a print manager. Its purpose is to make changes to device memory permanent until the device is turned off. It is usually only appropriate for error patches or to change the system defaults on a device. The value of *Password or the current password input by a user must precede this invocation.

***DeviceAdjustMatrix:** *"[ transformation matrix ]"*

This QuotedValue provides a device-specific transformation matrix to compensate for any anamorphic scaling or offset problems inherent in the underlying mechanical marking device. If the device has no such inherent problems, the value of *DeviceAdjustMatrix is the identity matrix [1 0 0 1 0 0], and the entire entry is omitted from the PPD file.

A system administrator might need to add *DeviceAdjustMatrix to a local customization file for a particular device to compensate for slight shrinkage or magnification caused by motor speeds, media thicknesses, and so on. See section 2.6 for information on local customization files.

*Note* *The *ImageableArea figures given in the PPD files will no longer be exactly accurate if the device matrix is adjusted. Bear in mind, if this field is changed, any operations sensitive to the page boundaries might have to be recomputed slightly, or the results might be off the page.*

† This keyword requires the *Password value to be supplied in front of the invocation.

## 5.23 Features Accessible Only Through Job Control Language

On some devices, certain features can be accessed only through a job control language (JCL), which is managed independently from the PostScript language interpreter. Keywords pertaining to such features are referred to throughout this document as "*JCL keywords". A typical job which accesses certain features via JCL code would contain these components in this order:

- the code from *JCLBegin, which starts the JCL job
- the code, if any, to change the desired feature, such as *JCLResolution or *JCLFrameBufferSize
- the code from *JCLToPSInterpreter, which invokes the PostScript interpreter
- the PostScript language job
- the code from *JCLEnd, which ends the job and returns the device to its idle state, awaiting further JCL commands.

***JCLBegin:** *"JCL"*

***JCLToPSInterpreter:** *"JCL"*

***JCLEnd:** *"JCL"*

These QuotedValues provide the JCL commands to bracket one or more PostScript language jobs into one printed document. The job is emitted in the order shown in the introduction to this section. If any of the *JCL- keywords are present in a PPD file, then these three keywords must all be present.

Here is an example of these keywords, using Hewlett Packard's PJL as the JCL:

```
*JCLBegin: "<1B>%-12345X@PJL JOB<0A>"
*JCLToPSInterpreter: "@PJL ENTER LANGUAGE = POSTSCRIPT <0A>"
*JCLEnd: "<1B>%-12345X@PJL EOJ<0A><1B>%-12345X"
```

***JCLOpenUI** *mainKeyword***: PickOne | PickMany | Boolean**

***JCLCloseUI**: *mainKeyword*

These keywords are identical to the *OpenUI/*CloseUI keywords (see section 5.5 for a description), except that they are used to enclose only *JCL keywords. Like keywords for other selectable features, JCL keywords affect the user interface, and as such must be presented to the user in a consistent fashion. All JCL keywords that provide the user with selectable features will be enclosed in the *JCLOpenUI/*JCLCloseUI keywords. If a print manager does

not wish to offer selection of features via JCL to the user, the parser can simply skip all sections of the PPD file that are bracketed by *JCLOpenUI/ *JCLCloseUI.

***JCLFrameBufferSize** *frameBufferOption*: *"JCL"*

This entry provides the JCL code to change the frame buffer size. Note that requesting a larger frame buffer size means that less memory is available for resources such as downloaded fonts.

Although the value looks like an InvocationValue, the *JCL keywords have special parsing rules and such values are treated like QuotedValues. This is because the values usually contain out-of-range byte codes in hexadecimal strings, which the print manager must translate before emitting into the job stream.

The values for *frameBufferOption* are device-specific. One of the options must be Off, with a corresponding QuotedValue that sends the JCL code to turn off the ability to set the frame buffer size. The results of this action are device-dependent. Other possibilities for options include any of the media size options supported by the device, with the corresponding JCL code requesting the frame buffer size appropriate for that media size. See section 5.9 for a description of media option keywords.

*Note* *On some devices, setting the frame buffer size may cause the device's memory to be reinitialized, removing anything that had previously been downloaded outside the server loop (at save level 0). For example, downloaded fonts, patterns, prologs, forms, and other downloaded resources would be removed from the device's memory.*

Here is an example of the frame buffer size keywords in a PPD file:

```
*JCLOpenUI *JCLFrameBufferSize/Frame Buffer Size: PickOne
*DefaultJCLFrameBufferSize: Letter
*OrderDependency: 20 JCLSetup *JCLFrameBufferSize
*JCLFrameBufferSize Off: '@PJL SET PAGEPROTECT = OFF<0A>'
*JCLFrameBufferSize Letter: '@PJL SET PAGEPROTECT = LTR<0A>'
*JCLFrameBufferSize Legal: '@PJL SET PAGEPROTECT = LGL<0A>'
*JCLCloseUI: *JCLFrameBufferSize
```

**F*DefaultJCLFrameBufferSize***:frameBufferOption*:

This indicates the default frame buffer size set by a JCL command. The value must match one of the options listed under *JCLFrameBufferSize.

***?JCLFrameBufferSize**: *"query"*

This query returns a string denoting the current frame buffer size set by a JCL command. The string returned must be a valid *frameBufferOption* listed under *JCLFrameBufferSize. If it is not possible to determine the frame buffer size, this query will be omitted.

***JCLResolution** *resolutionOption*: *"JCL"*

This entry provides the JCL code to change the resolution. There is one entry for each resolution supported by the device. For a complete explanation of *resolutionOption* and its possible values, see the description of *DefaultResolution in section 5.16.

Although the value looks like an InvocationValue, the *JCL keywords have special parsing rules and such values are treated like QuotedValues. This is because the values usually contain out-of-range byte codes in hexadecimal strings, which the print manager must translate before emitting into the job stream.

*Note* *On some devices, setting the resolution may cause the device's memory to be reinitialized, removing anything that had previously been downloaded outside the server loop (at save level 0). For example, downloaded fonts, patterns, prologs, forms, and other downloaded resources would be removed from the device's memory.*

***DefaultJCLResolution**: *resolutionOption* | Unknown

This indicates the default resolution set by a JCL command. The value must match one of the options listed under *JCLResolution. For a complete explanation of *resolutionOption*, see the description of *DefaultResolution in section 5.16.

***?JCLResolution**: *"query"*

This query returns a string denoting the device resolution set by a JCL command. The resolution returned must be a valid *resolutionOption* listed under *JCLResolution. If it is not possible to determine the resolution, this query will be omitted.

Here is a typical entry, using PJL as the JCL:

```
*JCLOpenUI *JCLResolution/Resolution Settings: PickOne
*DefaultJCLResolution: 300dpi
*OrderDependency: 10 JCLSetup *JCLResolution
*JCLResolution 300dpi/300 DPI: "@PJL SET RESOLUTION = 300<0A>"
*JCLResolution 600dpi/600 DPI: "@PJL SET RESOLUTION = 600<0A>"
*JCLCloseUI: *JCLResolution
```

# 6 Sample PPD File Structure

This section contains examples of PPD files for three types of devices

- a generic Level 1 black and white printer
- a generic Level 2 color printer
- a generic Level 1 imagesetter

## 6.1 Level 1 300 DPI Monochrome Printer

This PPD file describes a black and white duplex printer containing a PostScript Level 1 interpreter. The printer has a resolution of 300 spots per inch, and supports four levels of software-selectable resolution enhancement. The printer has multiple input and output trays, including an optional envelope feeder. It supports automatic tray switching, the Adobe Binary Communications Protocol, and contains a software-selectable Hewlett-Packard LaserJet emulator. It also supports Hewlett Packard's PJL job control language. It does not support the attachment of a writable file system, such as a hard disk.

```
*PPD-Adobe: "4.1"
*FormatVersion: "4.1"
*FileVersion: "1.0"
*LanguageEncoding: ISOLatin1
*LanguageVersion: English
*Product: "(Acme LaserPrinter II)"
*PSVersion: "(52.3) 1"
*ModelName: "Acme LaserPrinter Model II"
*ShortNickName: “Acme Laser Model 2 v52.3”
*NickName: "Acme LaserPrinter Model II v.52.3"
*PCFileName: “ACNIFTY1.PPD”

*%===== Installable Options =============
*OpenGroup: InstallableOptions/Options Installed
*OpenUI *Option1/Optional Envelope Feeder: Boolean
*DefaultOption1: False
*Option1 True/Installed: “”
*Option1 False/Not Installed: “”
*CloseUI: *Option1
*CloseGroup: InstallableOptions

*% ====== Constraints ===========
*% This device cannot print duplex on envelopes or transparencies,
*% It cannot output legal size paper to the rear output tray. It
*% cannot print from the envelope feeder unless the feeder is installed
*UIConstraints: *PageSize Envelope *Duplex
*UIConstraints: *Duplex *PageSize Envelope
*UIConstraints: *Duplex *MediaType Transparent
*UIConstraints: *MediaType Transparent *Duplex
*UIConstraints: *PageSize Legal *OutputBin Rear
*UIConstraints: *OutputBin Rear *PageSize Legal
*UIConstraints: *Option1 False *InputSlot Envelope
```

```
*%=== Basic Capabilities ===============
*LanguageLevel: 1
*DefaultColorSpace: Gray
*FreeVM: "1133530"
*Throughput: "8"

*Protocols: BCP PJL
*Emulators: hplj
*StartEmulator_hplj: "currentfile /hpcl statusdict /emulate get exec "
*StopEmulator_hplj: "<1B7F>0"
*JCLBegin: "<1B>%-12345X@PJL JOB<0A>"
*JCLToPSInterpreter: "@PJL ENTER LANGUAGE = POSTSCRIPT <0A>"
*JCLEnd: "<1B>%-12345X@PJL EOJ<0A><1B>%-12345X"

*Password: "0"
*ExitServer: " count 0 eq {  % is the password on the stack?
    true
  }{dup% potential password
    statusdict /checkpassword get exec not
  } ifelse
  {  %  if no password or not valid
    (WARNING : Cannot perform the exitserver command.) =
    (Password supplied is not valid.) =
    (Please contact the author of this software.) = flush quit
  } if
  serverdict /exitserver get exec"
*End

*Reset: "count 0 eq {  % is the password on the stack?
    true
  }{dup% potential password
    statusdict /checkpassword get exec not
  } ifelse
  {  %  if no password or not valid
    (WARNING : Cannot reset printer.) =
    (Password supplied is not valid.) =
    (Please contact the author of this software.) = flush quit
  } if
  serverdict /exitserver get exec
  systemdict /quit get exec
  (WARNING : Printer Reset Failed.) = flush"
*End

*%=== Resolution ======================
*DefaultResolution: 300dpi
*?Resolution: "save
  initgraphics
  0 0 moveto currentpoint matrix defaultmatrix transform
  0 72 lineto currentpoint matrix defaultmatrix transform
  3 -1 roll sub dup mul
  3 1 roll exch sub dup mul
  add sqrt round cvi
  (          ) cvs print (dpi) = flush
restore"
*End
```

```
*OpenUI *Smoothing/Resolution Enhancement Level: PickOne
*OrderDependency: 40 AnySetup *Smoothing
*DefaultSmoothing: Medium
*Smoothing None/Off: "0 statusdict /setdoret get exec"
*Smoothing Light: "1 statusdict /setdoret get exec"
*Smoothing Medium: "2 statusdict /setdoret get exec"
*Smoothing Dark: "3 statusdict /setdoret get exec"
*?Smoothing: "save
  [(None)(Light)(Medium)(Dark)]
  statusdict /doret {get exec}
  stopped { pop pop (Unknown)} if
  = flush restore"
*End
*CloseUI: *Smoothing

*% Halftone Information ===========================
*ScreenFreq: "60.0"
*ScreenAngle: "45.0"
*DefaultScreenProc: Dot
*ScreenProc Dot: "
{abs exch abs 2 copy add 1 gt {1 sub dup mul exch 1 sub dup mul add 1
sub}{dup mul exch dup mul add 1 exch sub}ifelse}
"
*End
*ScreenProc Line: "{pop}"
*ScreenProc Ellipse: "{dup 5 mul 8 div mul exch dup mul exch add
 sqrt 1 exch sub}
"
*End

*DefaultTransfer: Null
*Transfer Null: "{}"
*Transfer Null.Inverse: "{1 exch sub}"
```

```
*% === Paper Handling ====================
*% Use these entries to set paper size unless there is specific reason
*% to use PageRegion, such as when using manual feed.
*OpenUI *PageSize: PickOne
*OrderDependency: 20 AnySetup *PageSize
*DefaultPageSize: Unknown
*PageSize Letter: "statusdict /lettertray get exec"
*PageSize Legal: "statusdict /legaltray get exec"
*PageSize A4: "statusdict /a4tray get exec"
*PageSize Envelope.270.500/AcmeCorp Envelope: "
  statusdict/acmeenvelopetray get exec"
*End
*PageSize Comm10/Comm #10 Envelope: "
   statusdict /4.125x9.5envelopetray get exec"
*End
*?PageSize: "save
  8 dict
   dup /lettertray (Letter) put
   dup /legaltray (Legal) put
   dup /a4tray (A4) put
   dup /4.125x9.5envelopetray (Comm10) put
   dup /acmeenvelopetray (Envelope.270.500) put
  statusdict /papersize get exec
  3 1 roll  {get} stopped {(Unknown)}if
  exch  not { print (.Transverse) }if
  = flush
  restore
"
*End
*CloseUI: *PageSize

*% These entries set up the frame buffer. Usually used with manual feed.
*OpenUI *PageRegion: PickOne
*OrderDependency: 30 AnySetup *PageRegion
*DefaultPageRegion: Unknown
*PageRegion Letter: "letter"
*PageRegion Legal: "legal"
*PageRegion A4: "a4"
*PageRegion Envelope.270.500/AcmeCorp Envelope: "acmeenvelope"
*PageRegion Comm10/Comm #10 Envelope: "4.125x9.5envelope"
*CloseUI: *PageRegion

*% These entries provide the imageable areas of the media option keywords.
*DefaultImageableArea: Unknown
*ImageableArea Letter: "13.1 12.1 596.2 774.3 "
*ImageableArea Legal: "15.1 13.1 597.2 991.1 "
*ImageableArea A4: "16.1 14.3 583.2 823.6 "
*ImageableArea Envelope.270.500/AcmeCorp Envelope: "16.3 20.2 265.2 495.1 "
*ImageableArea Comm10/Comm #10 Envelope: "17.2 21.3 292.12 672.23"
*?ImageableArea: "save
  /cvp {(                ) cvs print ( ) print } bind def
  /upperright {10000 mul floor 10000 div} bind def
  /lowerleft {10000 mul ceiling 10000 div} bind def
  newpath clippath pathbbox
  4 -2 roll exch 2 {lowerleft cvp} repeat
  exch 2 {upperright cvp} repeat flush
restore"
*End
```

```
*% These provide the physical dimensions of the media, by option keyword.
*DefaultPaperDimension: Unknown
*PaperDimension Letter: "612 792"
*PaperDimension Legal: "612 1008"
*PaperDimension A4: "595 842"
*PaperDimension Envelope.270.500/AcmeCorp Envelope: "270 500"
*PaperDimension Comm10/Comm #10 Envelope: "297 684"

*OpenUI *MediaType: PickOne
*OrderDependency: 20 AnySetup *MediaType
*DefaultMediaType: Opaque
*MediaType Transparent: "1 statusdict /setmediatype get exec"
*MediaType Opaque: "0 statusdict /setmediatype get exec"
*?MediaType: "save
  [(Opaque)(Transparent)]
  statusdict /mediatype {get exec}
  stopped {pop pop (Unknown)}if = flush restore"
*End
*CloseUI: *MediaType

*OpenUI *InputSlot: PickOne
*OrderDependency: 15 AnySetup *InputSlot
*DefaultInputSlot: Upper
*InputSlot Upper: " 0 statusdict /setpapertray get exec"
*InputSlot Lower: " 1 statusdict /setpapertray get exec"
*InputSlot Envelope: " 2 statusdict /setpapertray get exec"
*?InputSlot: "
 save
  3 dict
   dup /0 (Upper) put
   dup /1 (Lower) put
   dup /2 (Envelope) put
   statusdict /papertray get exec
   {get} stopped {pop pop (Unknown)} if = flush
 restore"
*End
*CloseUI: *InputSlot


*OpenUI *ManualFeed: Boolean
*OrderDependency: 15 AnySetup *ManualFeed
*DefaultManualFeed: False
*ManualFeed True: "statusdict /manualfeed true put"
*ManualFeed False: "statusdict /manualfeed false put"
*?ManualFeed: "save
  statusdict /manualfeed get
  {(True)}{(False)}ifelse = flush restore"
*End
*CloseUI: *ManualFeed


*DefaultOutputOrder: Normal
*DefaultOutputBin: Upper
*PageStackOrder Upper: Normal
*PageStackOrder Rear: Reverse
```

```
*OpenUI *TraySwitch: Boolean
*OrderDependency: 30 AnySetup *TraySwitch
*DefaultTraySwitch: False
*TraySwitch True: "statusdict /trayswitch true put"
*TraySwitch False: "statusdict /trayswitch false put"
*?TraySwitch: "save
  statusdict /trayswitch get
  {(True)}{(False)}ifelse = flush restore"
*End
*CloseUI: *TraySwitch

*OpenUI *Duplex: PickOne
*OrderDependency: 30 AnySetup *Duplex
*DefaultDuplex: None
*Duplex DuplexTumble: "
  true statusdict /setduplexmode get exec
  true statusdict /settumble get exec"
*End
*Duplex DuplexNoTumble: "
  true statusdict /setduplexmode get exec
  false statusdict /settumble get exec"
*End
*Duplex None: "
  false statusdict /setduplexmode get exec
  false statusdict /settumble get exec"
*End

*?Duplex: "save
        statusdict /duplexmode get exec
        {tumble{(DuplexTumble)}{(DuplexNoTumble)}ifelse}
        {(None)}ifelse = flush restore"
*End
*CloseUI: *Duplex

*% Font Information =====================

*DefaultFont: Courier
*Font AvantGarde-Book: Standard "(001.002)" Standard ROM
*Font AvantGarde-BookOblique: Standard "(001.002)" Standard ROM
*Font AvantGarde-Demi: Standard "(001.003)" Standard ROM
*Font AvantGarde-DemiOblique: Standard "(001.003)" Standard ROM
*Font Courier: Standard "(002.002)" Standard ROM
*Font Courier-Bold: Standard "(002.002)" Standard ROM
*Font Courier-BoldOblique: Standard "(002.002)" Standard ROM
*Font Symbol: Special "(001.003)" Special ROM
*Font Times-Bold: Standard "(001.002)" Standard ROM
*Font Times-BoldItalic: Standard "(001.004)" Standard ROM
*Font Times-Italic: Standard "(001.002)" Standard ROM
*Font Times-Roman: Standard "(001.002)" Standard ROM
*Font ZapfDingbats: Special "(001.002)" Special ROM
```

```
*?FontQuery: "save
 /str 100 string dup 0 (fonts/) putinterval def
 {
   count 1 gt
   {
     exch dup str 6 94 getinterval cvs
     (/) print dup print (:) print exch
     FontDirectory exch known
     { pop (Yes) }
    {
      length 6 add str 0 3 -1 roll getinterval
      mark exch status
        {cleartomark (Yes)}{cleartomark (No)} ifelse
     } ifelse = flush
   }
   {exit} ifelse
 }bind loop
 (*) = flush
restore
"
*End

*?FontList: "save
  FontDirectory { pop == } bind forall flush
   (*) = flush
restore
"
*End

*% Printer Messages (verbatim from printer):
*Message: "%%[ exitserver: permanent state may be changed ]%%"
*Message: "\FontName\ not found, using Courier"


*% Status (format: %%[ status: <one of these> ]%% )
*Status: "idle"
*Status: "busy"
*Status: "waiting"
*Status: "printing"
*Status: "initializing"
*Status: "PrinterError: Optical System Error "
*Status: "PrinterError:  Main Motor Err "
*Status: "PrinterError: Prn not Rdy"/Error: Printer Not Ready

*% Input Sources (format:%%[status:<stat>;source:<one of these>]%%)
*Source: "serial9"
*Source: "serial25"
*Source: "AppleTalk"
*Source: "Centronics"

*% Printer Error (format: %%[ PrinterError: <one of these> ]%%)
*PrinterError: "Optical System Error "
*PrinterError: " Main Motor Err "
*PrinterError: "Prn not Rdy"/Error: Printer Not Ready
```

```
*% Color Separation Information ====================
*DefaultColorSep: ProcessBlack.60lpi.300dpi/60 lpi / 300 dpi

*InkName: ProcessBlack/Process Black
*InkName: CustomColor/Custom Color
*InkName: ProcessCyan/Process Cyan
*InkName: ProcessMagenta/Process Magenta
*InkName: ProcessYellow/Process Yellow

*%  For 60 lpi / 300 dpi  ==============================
*ColorSepScreenAngle ProcessBlack.60lpi.300dpi/60 lpi / 300 dpi: "45"
*ColorSepScreenAngle CustomColor.60lpi.300dpi/60 lpi / 300 dpi: "45"
*ColorSepScreenAngle ProcessCyan.60lpi.300dpi/60 lpi / 300 dpi: "15"
*ColorSepScreenAngle ProcessMagenta.60lpi.300dpi/60 lpi / 300 dpi: "75"
*ColorSepScreenAngle ProcessYellow.60lpi.300dpi/60 lpi / 300 dpi: "0"

*ColorSepScreenFreq ProcessBlack.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq CustomColor.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq ProcessCyan.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq ProcessMagenta.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq ProcessYellow.60lpi.300dpi/60 lpi / 300 dpi: "60"

*%  For 53 lpi / 300 dpi  ==============================

*ColorSepScreenAngle ProcessBlack.53lpi.300dpi/53 lpi / 300 dpi: "45.0"
*ColorSepScreenAngle CustomColor.53lpi.300dpi/53 lpi / 300 dpi: "45.0"
*ColorSepScreenAngle ProcessCyan.53lpi.300dpi/53 lpi / 300 dpi: "71.5651"
*ColorSepScreenAngle ProcessMagenta.53lpi.300dpi/53 lpi / 300 dpi: "18.43"
*ColorSepScreenAngle ProcessYellow.53lpi.300dpi/53 lpi / 300 dpi: "0.0"

*ColorSepScreenFreq ProcessBlack.53lpi.300dpi/53 lpi / 300 dpi: "53.033"
*ColorSepScreenFreq CustomColor.53lpi.300dpi/53 lpi / 300 dpi: "53.033"
*ColorSepScreenFreq ProcessCyan.53lpi.300dpi/53 lpi / 300 dpi: "47.43"
*ColorSepScreenFreq ProcessMagenta.53lpi.300dpi/53 lpi / 300 dpi: "47.43"
*ColorSepScreenFreq ProcessYellow.53lpi.300dpi/53 lpi / 300 dpi: "50.0"

*% end of PPD file for Acme LaserPrinter II
```

## 6.2 Level 2 Color Printer

This PPD file describes a Level 2 color printer with one input slot, one output bin, and two supported page sizes. The printer accepts manually fed pages, and can have a hard disk attached to it. It can also do color separations at the printer, when sent a composite color file.

```
*PPD-Adobe: "4.1"
*FormatVersion: "4.1"
*FileVersion: "1.0"
*LanguageEncoding: ISOLatin1
*LanguageVersion: English
*Product: "(Acme Color Printer)"
*PSVersion: "(2000.0) 0"
*ModelName: "Acme Color Printer v.2000"
*NickName: "Acme Color Printer v.2000"
*PCFileName: “ACCOLOR1.PPD”

*%=== Basic Capabilities ===============
*LanguageLevel: 2
*DefaultColorSpace: CMYK
*FreeVM: "8134935"
*FileSystem: True
*?FileSystem: "save statusdict /diskonline get exec
  {(True)}{(False)} ifelse = flush restore"
*End

*Throughput: "1"

*Password: "0"
*ExitServer: "
 count 0 eq
 { false } { true exch startjob } ifelse
  not { (WARNING: Cannot modify initial VM.) =
            (Missing or invalid password.) =
            (Please contact the author of this software.) = flush quit
  } if"
"
*End

*Reset: "
   count 0 eq
   { false } { true exch startjob } ifelse
   not { (WARNING: Cannot reset printer.) =
            (Missing or invalid password.) =
            (Please contact the author of this software.) = flush quit
          } if
   systemdict /quit get exec
   (WARNING : Printer Reset Failed.) = flush
"
*End
```

```
*%=== Built-In Color Rendering Dictionaries ===========
*ColorRenderDict 1/Copy Paper: “
/OEMDict1 /ColorRendering findresource setcolorrendering"
*End
*ColorRenderDict 2/Bond Paper: “
/OEMDict2 /ColorRendering findresource setcolorrendering"
*End
*ColorRenderDict 3/Transparency: “
/OEMDict3 /ColorRendering findresource setcolorrendering"
*End

*%=== Resolution Information ======================
*DefaultResolution: 300dpi
*?Resolution: "save
  currentpagedevice /HWResolution get
  0 get
  (      ) cvs print
  (dpi) = flush
  restore
"
*End

*% Halftone Information ===========================
*ScreenFreq: "60.0"
*ScreenAngle: "45.0"
*DefaultScreenProc: Dot
*ScreenProc Dot: "
{abs exch abs 2 copy add 1 gt {1 sub dup mul exch 1 sub dup mul add 1
sub}{dup mul exch dup mul add 1 exch sub}ifelse}
"
*End
*ScreenProc Line: "{pop}"
*ScreenProc Ellipse: "{dup 5 mul 8 div mul exch dup mul exch add
 sqrt 1 exch sub}
"
*End

*DefaultTransfer: Null
*Transfer Null: "{}"
*Transfer Null.Inverse: "{1 exch sub}"

*% Paper Handling ==================
*% Use these entries to set paper size, unless there is
*% specific reason to use PageRegion, such as with manual feed.
*OpenUI *PageSize: PickOne
*OrderDependency: 20 AnySetup *PageSize
*PageSize Letter: "1 dict dup /PageSize [612 792] put setpagedevice"
*PageSize Legal: "1 dict dup /PageSize [612 1008] put setpagedevice"
*DefaultPageSize: Letter
```

```
*?PageSize: "save
   currentpagedevice /PageSize get aload pop
   2 copy gt {exch} if (Unknown)
   2 dict
   dup [612 792] (Letter) put
   dup [612 1008] (Legal) put
   { exch aload pop 4 index sub abs 5 le exch 5 index sub abs 5 le and
     { exch pop exit } {pop} ifelse
   } bind forall = flush pop pop
 restore
"
*End
*CloseUI: *PageSize
*% These entries set up the frame buffer. Usually used with manual feed.
*OpenUI *PageRegion: PickOne
*OrderDependency: 30 AnySetup *PageRegion
*PageRegion Letter: "1 dict dup /PageSize [612 792] put setpagedevice"
*PageRegion Legal: "1 dict dup /PageSize [612 1008] put setpagedevice"
*DefaultPageRegion: Letter
*CloseUI: *PageRegion

*% The following entries provide information about specific paper keywords.

*DefaultImageableArea: Letter
*ImageableArea Letter: "13 12 596 774 "
*ImageableArea Legal: "15 13 597 991 "
*?ImageableArea: " save /cvp { cvi (            ) cvs
  print ( ) print } bind def
  newpath clippath pathbbox
  4 -2 roll exch 2 {ceiling cvp} repeat
  exch 2 {floor cvp} repeat flush
 restore
"
*End

*% These provide the physical dimensions of the paper (by keyword)
*DefaultPaperDimension: Letter
*PaperDimension Letter: "612 792"
*PaperDimension Legal: "612 1008"

*OpenUI *ManualFeed: Boolean
*OrderDependency: 15 AnySetup *ManualFeed
*ManualFeed True: "1 dict dup /ManualFeed true put setpagedevice"
*ManualFeed False: "1 dict dup /ManualFeed false put setpagedevice"
*DefaultManualFeed: False
*?ManualFeed: "save
   currentpagedevice /ManualFeed get
   {(True)}{(False)}ifelse
   = flush
  restore"
*End
*CloseUI: *ManualFeed
```

```
*% Font Information =====================

*DefaultFont: Courier
*Font AvantGarde-Book: Standard "(001.002)" Standard ROM
*Font AvantGarde-BookOblique: Standard "(001.002)" Standard ROM
*Font AvantGarde-Demi: Standard "(001.003)" Standard ROM
*Font AvantGarde-DemiOblique: Standard "(001.003)" Standard ROM
*Font Courier: Standard "(002.002)" Standard ROM
*Font Courier-Bold: Standard "(002.002)" Standard ROM
*Font Courier-BoldOblique: Standard "(002.002)" Standard ROM
*Font Symbol: Special "(001.003)" Special ROM
*Font Times-Bold: Standard "(001.002)" Standard ROM
*Font Times-BoldItalic: Standard "(001.004)" Standard ROM
*Font Times-Italic: Standard "(001.002)" Standard ROM
*Font Times-Roman: Standard "(001.002)" Standard ROM
*Font ZapfDingbats: Special "(001.002)" Special ROM

*?FontQuery: "
   save
   { count 1 gt
      { exch dup 127 string cvs (/) print print (:) print
       /Font resourcestatus {pop pop (Yes)} {(No)} ifelse =
      } { exit } ifelse
   } bind loop
   (*) = flush
   restore"
*End

*?FontList: "
   save (*) {cvn ==} 128 string /Font resourceforall
   (*) = flush restore"
*End


*% Printer Messages (verbatim from printer):
*Message: "%%[ exitserver: permanent state may be changed ]%%"
*Message: "\FontName\ not found, using Courier"


*% Status (format: %%[ status: <one of these> ]%% )
*Status: "idle"
*Status: "busy"
*Status: "waiting"
*Status: "printing"
*Status: "initializing"
*Status: "PrinterError: "Optical System Error "
*Status: "PrinterError: " Cover Open "
*Status: "PrinterError: "Prnter Wrmng"/PrinterError: Printer Warming Up


*% Input Sources (format:%%[status:<stat>;source:<one of these>]%%)
*Source: "%Serial%"
*Source: "%SerialB%"
*Source: "%LocalTalk%"
*Source: "%Parallel%"
```

```
*% Printer Error (format: %%[ PrinterError: <one of these> ]%%)
*PrinterError: "Optical System Error "
*PrinterError: " Cover Open "
*PrinterError: "Prnter Wrmng"/Printer Warming Up

*% Color Separation Information =====================

*OpenUI *Separations: Boolean
*OrderDependency: 40 AnySetup *Separations
*Separations True: "1 dict dup /Separations true put setpagedevice"
*Separations False: "1 dict dup /Separations false put setpagedevice"
*DefaultSeparations: False
*?Separations: "save currentpagedevice /Separations get
  {(True)}{(False)}ifelse = flush restore
"
*End
*CloseUI: *Separations

*DefaultColorSep: ProcessBlack.60lpi.300dpi/60 lpi / 300 dpi

*InkName: ProcessBlack/Process Black
*InkName: CustomColor/Custom Color
*InkName: ProcessCyan/Process Cyan
*InkName: ProcessMagenta/Process Magenta
*InkName: ProcessYellow/Process Yellow

*%  For 60 lpi / 300 dpi  ==============================

*ColorSepScreenAngle ProcessBlack.60lpi.300dpi/60 lpi / 300 dpi: "45"
*ColorSepScreenAngle CustomColor.60lpi.300dpi/60 lpi / 300 dpi: "45"
*ColorSepScreenAngle ProcessCyan.60lpi.300dpi/60 lpi / 300 dpi: "15"
*ColorSepScreenAngle ProcessMagenta.60lpi.300dpi/60 lpi / 300 dpi: "75"
*ColorSepScreenAngle ProcessYellow.60lpi.300dpi/60 lpi / 300 dpi: "0"

*ColorSepScreenFreq ProcessBlack.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq CustomColor.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq ProcessCyan.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq ProcessMagenta.60lpi.300dpi/60 lpi / 300 dpi: "60"
*ColorSepScreenFreq ProcessYellow.60lpi.300dpi/60 lpi / 300 dpi: "60"

*%  For 53 lpi / 300 dpi  ==============================

*ColorSepScreenAngle ProcessBlack.53lpi.300dpi/53 lpi / 300 dpi: "45.0"
*ColorSepScreenAngle CustomColor.53lpi.300dpi/53 lpi / 300 dpi: "45.0"
*ColorSepScreenAngle ProcessCyan.53lpi.300dpi/53 lpi / 300 dpi: "71.5651"
*ColorSepScreenAngle ProcessMagenta.53lpi.300dpi/53 lpi / 300 dpi: "18.43"
*ColorSepScreenAngle ProcessYellow.53lpi.300dpi/53 lpi / 300 dpi: "0.0"

*ColorSepScreenFreq ProcessBlack.53lpi.300dpi/53 lpi / 300 dpi: "53.033"
*ColorSepScreenFreq CustomColor.53lpi.300dpi/53 lpi / 300 dpi: "53.033"
*ColorSepScreenFreq ProcessCyan.53lpi.300dpi/53 lpi / 300 dpi: "47.43"
*ColorSepScreenFreq ProcessMagenta.53lpi.300dpi/53 lpi / 300 dpi: "47.43"
*ColorSepScreenFreq ProcessYellow.53lpi.300dpi/53 lpi / 300 dpi: "50.0"

*% end of PPD file for Acme Color Printer
```

### 6.3 Level 1 Imagesetter

This paragraph describes a generic Level 1 imagesetter. This is a roll-fed device, and it allows the user to invoke a custom page size. It supports variable resolution and accurate screens. It supports the features mirror print and negative print, which have been grouped together by the *OpenGroup/ *CloseGroup keyword pair. This device ships with several fonts built into the ROM of the device, and the font Avant-Garde on a separate hard disk.

```
*PPD-Adobe: "4.1"
*FormatVersion: "4.1"
*FileVersion: "1.0"
*LanguageEncoding: ISOLatin1
*LanguageVersion: English
*Product: "(Acme Imagesetter)"
*PSVersion: "(52.3) 7"
*ModelName: "Acme Imagesetter v.52.3"
*NickName: "Acme Imagesetter v.52.3"
*PCFileName: “ACIMAGE1.PPD”


*%=== Basic Capabilities ===============

*LanguageLevel: 1
*Extensions: CMYK FileSystem
*ColorDevice: False
*DefaultColorSpace: Gray
*VariablePaperSize: True
*FreeVM: "8134935"

*FileSystem: True
*?FileSystem: "save statusdict /diskonline get exec
   {(True)}{(False)} ifelse = flush restore"
*End

*Throughput: "1"

*Password: "0"

*ExitServer: " count 0 eq {  % is the password on the stack?
    true
  }{dup% potential password
    statusdict /checkpassword get exec not
  } ifelse
  {  %  if no password or not valid
    (WARNING : Cannot perform the exitserver command.) =
    (Password supplied is not valid.) =
    (Please contact the author of this software.) = flush quit
  } if
  serverdict /exitserver get exec"
*End
```

```
*Reset: "count 0 eq {  % is the password on the stack?
    true
  }{dup% potential password
    statusdict /checkpassword get exec not
  } ifelse
  {  %  if no password or not valid
    (WARNING : Cannot reset printer.) =
    (Password supplied is not valid.) =
    (Please contact the author of this software.) = flush quit
  } if
  serverdict /exitserver get exec
  systemdict /quit get exec
  (WARNING : Printer Reset Failed.) = flush"
*End

*%=== Resolution Information ==================

*OpenUI *Resolution/Choose Resolution: PickOne
*OrderDependency: 10 AnySetup *Resolution
*Resolution 600dpi: "600 statusdict /setresolution get exec"
*Resolution 1200dpi: "1200 statusdict /setresolution get exec"
*Resolution 2400dpi: "2400 statusdict /setresolution get exec"
*DefaultResolution: 1200dpi
*?Resolution: "save
  initgraphics
  0 0 moveto currentpoint matrix defaultmatrix transform
  0 72 lineto currentpoint matrix defaultmatrix transform
  3 -1 roll sub dup mul
  3 1 roll exch sub dup mul
  add sqrt round cvi
  (          ) cvs print (dpi) = flush
restore
"
*End
*CloseUI: *Resolution

*% === Halftone Information ==========================

*ScreenFreq: "60.0"
*ScreenAngle: "45.0"
*DefaultScreenProc: Dot
*ScreenProc Dot: "
{abs exch abs 2 copy add 1 gt {1 sub dup mul exch 1 sub dup mul add 1
sub}{dup mul exch dup mul add 1 exch sub}ifelse}
"
*End
*ScreenProc Line: "{pop}"
*ScreenProc Ellipse: "{dup 5 mul 8 div mul exch dup mul exch add
 sqrt 1 exch sub}
"
*End

*AccurateScreensSupport: True

*DefaultTransfer: Null
*Transfer Null: "{}"
*Transfer Null.Inverse: "{1 exch sub}"
```

```
*% Paper Handling ====================

*% Use these entries to set paper size most of the time, unless there is
*% specific reason to use PageRegion or PaperTray.
*OpenUI *PageSize: PickOne
*OrderDependency: 30 AnySetup *PageSize
*DefaultPageSize: Letter
*PageSize Letter: "letter"
*PageSize Legal: "legal"
*PageSize Ledger: "ledger"
*PageSize Tabloid: "11x17"
*?PageSize: "save
  mark
  statusdict /pageparams get exec
  pop pop  % margins and orientation
  4 dict
     dup [792 612] (Letter) put
     dup [1008 612] (Legal) put
     dup [1224 792] (Tabloid) put
     dup [792 1224] (Ledger) put
  (Unknown) exch
  { exch aload pop
    4 index eq exch 5 index eq and
    { exch pop exit } { pop } ifelse
  } bind forall = flush cleartomark
restore
"
*End
*CloseUI: *PageSize

*% These entries set up the frame buffer. Same as *PageSize for an
*% imagesetter, which has no input trays or manual feed slot.
*OpenUI *PageRegion: PickOne
*OrderDependency: 40 AnySetup *PageRegion
*DefaultPageRegion: Letter
*PageRegion Letter: "letter"
*PageRegion Legal: "legal"
*PageRegion Ledger: "ledger"
*PageRegion Tabloid: "11x17"
*CloseUI: *PageRegion

*% These entries provide the imageable area for specific paper keywords.
*DefaultImageableArea: Letter
*ImageableArea Letter: "0.0 0.0 612.0 792.0"
*ImageableArea Legal: "0.0 0.0 612.0 1008.0"
*ImageableArea Ledger: "0.0 0.0 1224.0 792.0"
*ImageableArea Tabloid: "0.0 0.0 792.0 1224.0"
*?ImageableArea: "
save
  /cvp { (                ) cvs print ( ) print } bind def
  /upperright {10000 mul floor 10000 div} bind def
  /lowerleft {10000 mul ceiling 10000 div} bind def
  newpath clippath pathbbox
  4 -2 roll exch 2 {lowerleft cvp} repeat
  exch 2 {upperright cvp} repeat flush
 restore
"
*End
```

```
*% These provide the physical dimensions of the page (by option keyword)
*DefaultPaperDimension: Letter
*PaperDimension Letter: "612 792"
*PaperDimension Legal: "612 1008"
*PaperDimension Ledger: "1224 792"
*PaperDimension Tabloid: "792 1224"

*%=== Custom Page Sizes ==================================
*% These entries provide the code and parameter ranges for a user to set up custom page sizes.
*CustomPageSize True: "exch pop statusdict /setpageparams get exec"
*ParamCustomPageSize Width:  1 points 1 612
*ParamCustomPageSize Height:  2 points 1 1224
*ParamCustomPageSize WidthOffset:  3 points 0 512
*ParamCustomPageSize HeightOffset:  4 points 0 0
*ParamCustomPageSize Orientation:  5 int 0 1
*CenterRegistered: False
*?CurrentMediaWidth: "save statusdict /mediawidth get exec = flush restore"

*% === Imagesetter Information ============================
*OpenGroup: Imagesetter
*OpenUI *MirrorPrint/Mirror Print: Boolean
*OrderDependency: 40 AnySetup *MirrorPrint
*MirrorPrint True: "true statusdict /setmirrorprint get exec"
*MirrorPrint False: "false statusdict /setmirrorprint get exec"
*DefaultMirrorPrint: False
*?MirrorPrint: "save statusdict /mirrorprint get exec
  {(True)}{(False)}ifelse = flush restore"
*End
*CloseUI: *MirrorPrint

*OpenUI *NegativePrint/Negative Print: Boolean
*OrderDependency: 40 AnySetup *NegativePrint
*NegativePrint True: "true statusdict /setnegativeprint get exec"
*NegativePrint False: "false statusdict /setnegativeprint get exec"
*DefaultNegativePrint: False
*?NegativePrint: "save statusdict /negativeprint get exec
   {(True)}{(False)}ifelse = flush restore"
*End
*CloseUI: *NegativePrint
*CloseGroup: Imagesetter

*% Font Information =====================
*% For example purposes, this device ships with several fonts built into
*% the ROM of the device, and Avant-Garde on a separate hard disk
*DefaultFont: Courier
*Font AvantGarde-Book: Standard "(001.002)" Standard Disk
*Font AvantGarde-BookOblique: Standard "(001.002)" Standard Disk
*Font AvantGarde-Demi: Standard "(001.003)" Standard Disk
*Font AvantGarde-DemiOblique: Standard "(001.003)" Standard Disk
*Font Courier: Standard "(002.002)" Standard ROM
*Font Courier-Bold: Standard "(002.002)" Standard ROM
*Font Courier-BoldOblique: Standard "(002.002)" Standard ROM
*Font Symbol: Special "(001.003)" Special ROM
*Font Times-Bold: Standard "(001.002)" Standard ROM
*Font Times-BoldItalic: Standard "(001.004)" Standard ROM
*Font Times-Italic: Standard "(001.002)" Standard ROM
*Font Times-Roman: Standard "(001.002)" Standard ROM
*?FontQuery: "
```

```
save
 /str 100 string dup 0 (fonts/) putinterval def
 {
   count 1 gt
   {
     exch dup str 6 94 getinterval cvs
     (/) print dup print (:) print exch
     FontDirectory exch known
     { pop (Yes) }
    {
      length 6 add str 0 3 -1 roll getinterval
      mark exch status
        {cleartomark (Yes)}{cleartomark (No)} ifelse
     } ifelse = flush
   }
   {exit} ifelse
 }bind loop
 (*) = flush
restore
"
*End

*?FontList: "
save
  FontDirectory { pop == } bind forall flush
  /filenameforall where
  { pop (fonts/*)
   { dup length 6 sub 6 exch getinterval cvn == } bind
    128 string filenameforall flush
  } if
  (*) = flush
restore
"
*End

*% Printer Messages (verbatim from printer):
*Message: "%%[ exitserver: permanent state may be changed ]%%"
*Message: "\FontName\ not found, using Courier"

*% Status (format: %%[ status: <one of these> ]%% )
*Status: "idle"
*Status: "busy"
*Status: "waiting"
*Status: "printing"
*Status: "initializing"
*Status: "PrinterError: Cassette not loaded"
*Status: "PrinterError: Film Unit Error"

*% Input Sources (format:%%[status:<stat>;source:<one of these>]%%)
*Source: "Appletalk"
*Source: "Centronics"
*Source: "serial9"
*Source: "serial25"

*% Printer Error (format: %%[ PrinterError: <one of these> ]%%)
*PrinterError: "Cassette not loaded"
*PrinterError: "Film Unit Error"
*% Color Separation Information ======================
```

```
*DefaultColorSep: ProcessBlack.90lpi.1200dpi/90 lpi / 1200 dpi

*InkName: ProcessBlack/Process Black
*InkName: CustomColor/Custom Color
*InkName: ProcessCyan/Process Cyan
*InkName: ProcessMagenta/Process Magenta
*InkName: ProcessYellow/Process Yellow

*% For 90 lpi / 1200 dpi  ==============================

*ColorSepScreenAngle ProcessCyan.90lpi.1200dpi/90 lpi / 1200 dpi: "71.565"
*ColorSepScreenAngle ProcessMagenta.90lpi.1200dpi/90 lpi/1200 dpi: "18.43"
*ColorSepScreenAngle ProcessYellow.90lpi.1200dpi/90 lpi / 1200 dpi: "0"
*ColorSepScreenAngle ProcessBlack.90lpi.1200dpi/90 lpi / 1200 dpi: "45"
*ColorSepScreenAngle CustomColor.90lpi.1200dpi/90 lpi / 1200 dpi: "45"

*ColorSepScreenFreq ProcessCyan.90lpi.1200dpi/90 lpi / 1200 dpi: "94.8683"
*ColorSepScreenFreq ProcessMagenta.90lpi.1200dpi/90 lpi/1200 dpi: "94.86"
*ColorSepScreenFreq ProcessYellow.90lpi.1200dpi/90 lpi / 1200 dpi: "30"
*ColorSepScreenFreq ProcessBlack.90lpi.1200dpi/90 lpi / 1200 dpi: "84.852"
*ColorSepScreenFreq CustomColor.90lpi.1200dpi/90 lpi / 1200 dpi: "84.8528"

*ColorSepScreenProc ProcessYellow.90lpi.1200dpi/90 lpi / 1200 dpi: "
{2 {1 add 2 div 3 mul dup floor sub 2 mul 1 sub exch } repeat
abs exch abs 2 copy add 1 gt {1 sub dup mul exch 1 sub dup mul add 1
sub }{dup mul exch dup mul add 1 exch sub }ifelse }"
*End

*% For 110 lpi / 1200 dpi  ==============================

*ColorSepScreenAngle ProcessCyan.110lpi.1200dpi/110 lpi /1200 dpi: "70.01"
*ColorSepScreenAngle ProcessMagenta.110lpi.1200dpi/110 lpi/1200 dpi: "19"
*ColorSepScreenAngle ProcessYellow.110lpi.1200dpi/110 lpi / 1200 dpi: "0"
*ColorSepScreenAngle ProcessBlack.110lpi.1200dpi/110 lpi / 1200 dpi: "45"
*ColorSepScreenAngle CustomColor.110lpi.1200dpi/110 lpi / 1200 dpi: "45"

*ColorSepScreenFreq ProcessCyan.110lpi.1200dpi/110 lpi /1200 dpi: "102.52"
*ColorSepScreenFreq ProcessMagenta.110lpi.1200dpi/110 lpi/1200 dpi: "102"
*ColorSepScreenFreq ProcessYellow.110lpi.1200dpi/110 lpi/1200 dpi: "109.1"
*ColorSepScreenFreq ProcessBlack.110lpi.1200dpi/110 lpi/1200 dpi: "121.22"
*ColorSepScreenFreq CustomColor.110lpi.1200dpi/110 lpi/1200 dpi: "121.218"

*% For 90 lpi / 2400 dpi  ==============================

*ColorSepScreenAngle ProcessCyan.90lpi.2400dpi/90 lpi /2400 dpi: "71.5651"
*ColorSepScreenAngle ProcessMagenta.90lpi.2400dpi/90 lpi/2400 dpi: "18.44"
*ColorSepScreenAngle ProcessYellow.90lpi.2400dpi/90 lpi / 2400 dpi: "0"
*ColorSepScreenAngle ProcessBlack.90lpi.2400dpi/90 lpi / 2400 dpi: "45"
*ColorSepScreenAngle CustomColor.90lpi.2400dpi/90 lpi / 2400 dpi: "45"

*ColorSepScreenFreq ProcessCyan.90lpi.2400dpi/90 lpi / 2400 dpi: "94.8683"
*ColorSepScreenFreq ProcessMagenta.90lpi.2400dpi/90 lpi /2400 dpi: "94.87"
*ColorSepScreenFreq ProcessYellow.90lpi.2400dpi/90 lpi / 2400 dpi: "30"
*ColorSepScreenFreq ProcessBlack.90lpi.2400dpi/90 lpi /2400 dpi: "84.8528"
*ColorSepScreenFreq CustomColor.90lpi.2400dpi/90 lpi / 2400 dpi: "84.8528"

*ColorSepScreenProc ProcessYellow.90lpi.2400dpi/90 lpi / 2400 dpi: "
```

```
  {2 {1 add 2 div 3 mul dup floor sub 2 mul 1 sub exch } repeat
  abs exch abs 2 copy add 1 gt {1 sub dup mul exch 1 sub dup mul add 1
  sub }{dup mul exch dup mul add 1 exch sub }ifelse }"
*End

*% For 115 lpi / 2400 dpi  ==============================

*ColorSepScreenAngle ProcessCyan.115lpi.2400dpi/115 lpi /2400 dpi: "71.56"
*ColorSepScreenAngle ProcessMagenta.115lpi.2400dpi/115 lpi/ 2400 dpi: "18"
*ColorSepScreenAngle ProcessYellow.115lpi.2400dpi/115 lpi / 2400 dpi: "0"
*ColorSepScreenAngle ProcessBlack.115lpi.2400dpi/115 lpi / 2400 dpi: "45"
*ColorSepScreenAngle CustomColor.115lpi.2400dpi/115 lpi / 2400 dpi: "45"

*ColorSepScreenFreq ProcessCyan.115lpi.2400dpi/115 lpi/2400 dpi: "126.491"
*ColorSepScreenFreq ProcessMagenta.115lpi.2400dpi/115 lpi/ 2400 dpi: "126"
*ColorSepScreenFreq ProcessYellow.115lpi.2400dpi/115 lpi / 2400 dpi: "120"
*ColorSepScreenFreq ProcessBlack.115lpi.2400dpi/115 lpi/2400 dpi: "113.13"
*ColorSepScreenFreq CustomColor.115lpi.2400dpi/115 lpi/2400 dpi: "113.137"

*% For 130 lpi / 2400 dpi  ==============================

*ColorSepScreenAngle ProcessCyan.130lpi.2400dpi/130 lpi/2400 dpi: "71.565"
*ColorSepScreenAngle ProcessMagenta.130lpi.2400dpi/130 lpi/ 2400 dpi: "18"
*ColorSepScreenAngle ProcessYellow.130lpi.2400dpi/130 lpi / 2400 dpi: "0"
*ColorSepScreenAngle ProcessBlack.130lpi.2400dpi/130 lpi / 2400 dpi: "45"
*ColorSepScreenAngle CustomColor.130lpi.2400dpi/130 lpi / 2400 dpi: "45"

*ColorSepScreenFreq ProcessCyan.130lpi.2400dpi/130 lpi/2400 dpi: "126.491"
*ColorSepScreenFreq ProcessMagenta.130lpi.2400dpi/130 lpi/ 2400 dpi: "126"
*ColorSepScreenFreq ProcessYellow.130lpi.2400dpi/130 lpi/2400 dpi: "133.3"
*ColorSepScreenFreq ProcessBlack.130lpi.2400dpi/130 lpi/2400 dpi: "141.42"
*ColorSepScreenFreq CustomColor.130lpi.2400dpi/130 lpi/2400 dpi: "141.421"
*% end of PPD file for Acme Imagesetter
```

# Appendix A: User Interface Keywords

This appendix provides a list of keywords that are typically bracketed by the *OpenUI/*CloseUI keywords in PPD files produced by Adobe Systems. Only the main keywords are listed here; naturally, their associated defaults and queries would also be included in the *OpenUI/*CloseUI bracketing. Other keywords may also be bracketed by *OpenUI/*CloseUI; this list provides only the *typical* set.

*AccurateScreensSupport
*AdvanceMedia
*BindColor
*BindEdge
*BindType
*BindWhen
*BitsPerPixel
*BlackSubstitution
*Booklet
*Collate
*ColorModel
*CutMedia
*Duplex
*FoldType
*FoldWhen
*InputSlot
*InsertSheet
*Jog
*ManualFeed
*MediaColor
*MediaType
*MediaWeight
*MirrorPrint
*NegativePrint
*OutputBin
*OutputMode
*OutputOrder
*PageSize
*PageRegion
*Resolution
*Separations
*Signature
*Slipsheet
*Smoothing
*Sorter
*StapleLocation
*StapleOrientation
*StapleWhen
*StapleX
*StapleY
*TraySwitch

# Appendix B: Repeated Keywords

In the general model, if a main keyword, or specific combination of main and option keyword, is repeated within a PPD file or in an included PPD file, the first occurrence has precedence and future occurrences are ignored. For historical reasons, there are certain keywords in a PPD file that do not conform to the general model; specific main keywords are repeated, but all occurrences are relevant and should be recorded by a parser because their values are unique. For backward compatibility, the form of these keywords cannot be changed.

To provide assistance to PPD file parsers, the following is a list of main keywords (excluding structure keywords) which do not have option keywords to distinguish one instance from another, yet all instances are relevant, so the all occurrences of this main keyword and its associated unique values should be recorded by the PPD file parser.

*Emulators
*Extensions
*FaxSupport
*Include
*Message
*PrinterError
*Product
*Protocols
*PSVersion
*Source
*Status
*UIConstraints

# Appendix C: Character Encodings

The *LanguageEncoding keyword defines the encoding used by translation strings and certain QuotedValues in a PPD file. This appendix describes three encodings commonly used in PPD files, and how to convert between them. The three encoding options compared in this appendix are ISOLatin1, WindowsANSI, and MacStandard. ISOLatin1 encoding is commonly used in the Unix environment. WindowsANSI is defined by Microsoft for use in the Windows operating system. MacStandard is the encoding used by Macintosh computers.

Document managers will need to convert certain strings from the encoding used in the PPD file to the encoding used on their operating system. For document managers operating in the Macintosh, Windows, and Unix environments, this often means a conversion between two of the three encodings listed here. These tables are intended to help in that conversion.

Table 7 shows the three encoding vectors in their entirety. It is indexed by character code and contains the union of all of the characters in all three encoding vectors.

Tables 8, 9, and 10 contain only the differences between the three encoding vectors, and could be the basis for conversion tables in a document manager. Table 8 is indexed by the character code and name of each character in the WindowsANSI encoding vector. Table 9 is indexed by the character code and name of each character in the MacStandard encoding vector. Table 10 is indexed by the character code and name of each character in the ISOLatin1 encoding vector.

## C.1 All Encodings Indexed By Byte Code

Table 7 shows the three encoding vectors in their entirety. The first column gives the byte code. The second, third, and fourth columns give the PostScript language name of the character encoded at that position in the specified encoding vector. The word “unused” in a column means there is no printable character at that byte code position in the specified encoding vector.

**Table 8: All Encodings Indexed By Byte Code**

| *Code* | *WindowsANSI* | *ISOLatin1* | *MacStandard* |
|---|---|---|---|
| **0-31** | *unused* | *unused* | *unused* |
| **32** | space | space | space |
| **33** | exclam | exclam | exclam |
| **34** | quotedbl | quotedbl | quotedbl |
| **35** | numbersign | numbersign | numbersign |
| **36** | dollar | dollar | dollar |
| **37** | percent | percent | percent |
| **38** | ampersand | ampersand | ampersand |
| **39** | quotesingle | quoteright | quotesingle |
| **40** | parenleft | parenleft | parenleft |
| **41** | parenright | parenright | parenright |
| **42** | asterisk | asterisk | asterisk |
| **43** | plus | plus | plus |
| **44** | comma | comma | comma |
| **45** | hyphen | minus | hyphen |
| **46** | period | period | period |
| **47** | slash | slash | slash |
| **48** | zero | zero | zero |
| **49** | one | one | one |
| **50** | two | two | two |
| **51** | three | three | three |
| **52** | four | four | four |
| **53** | five | five | five |
| **54** | six | six | six |
| **55** | seven | seven | seven |
| **56** | eight | eight | eight |
| **57** | nine | nine | nine |
| **58** | colon | colon | colon |
| **59** | semicolon | semicolon | semicolon |
| **60** | less | less | less |
| **61** | equal | equal | equal |
| **62** | greater | greater | greater |
| **63** | question | question | question |
| **64** | at | at | at |
| **65-90** | A-Z | A-Z | A-Z |
| **91** | bracketleft | bracketleft | bracketleft |
| **92** | backslash | backslash | backslash |
| **93** | bracketright | bracketright | bracketright |
| **94** | asciicircum | asciicircum | asciicircum |

**Table 8: All Encodings Indexed By Byte Code**

| *Code* | *WindowsANSI* | *ISOLatin1* | *MacStandard* |
|---|---|---|---|
| **95** | underscore | underscore | underscore |
| **96** | grave | quoteleft | grave |
| **97-122** | a-z | a-z | a-z |
| **123** | braceleft | braceleft | braceleft |
| **124** | bar | bar | bar |
| **125** | braceright | braceright | braceright |
| **126** | asciitilde | asciitilde | asciitilde |
| **127** | *unused* | *unused* | *unused* |
| **128** | *unused* | *unused* | Adieresis |
| **129** | *unused* | *unused* | Aring |
| **130** | quotesinglbase | *unused* | Ccedilla |
| **131** | florin | *unused* | Eacute |
| **132** | quotedblbase | *unused* | Ntilde |
| **133** | ellipsis | *unused* | Odieresis |
| **134** | dagger | *unused* | Udieresis |
| **135** | daggerdbl | *unused* | aacute |
| **136** | circumflex | *unused* | agrave |
| **137** | perthousand | *unused* | acircumflex |
| **138** | Scaron | *unused* | adieresis |
| **139** | guilsinglleft | *unused* | atilde |
| **140** | OE | *unused* | aring |
| **141** | *unused* | *unused* | ccedilla |
| **142** | *unused* | *unused* | eacute |
| **143** | *unused* | *unused* | egrave |
| **144** | *unused* | dotlessi | ecircumflex |
| **145** | quoteleft | grave | edieresis |
| **146** | quoteright | acute | iacute |
| **147** | quotedblleft | circumflex | igrave |
| **148** | quotedblright | tilde | icircumflex |
| **149** | bullet | macron | idieresis |
| **150** | endash | breve | ntilde |
| **151** | emdash | dotaccent | oacute |
| **152** | tilde | dieresis | ograve |
| **153** | trademark | *unused* | ocircumflex |
| **154** | scaron | ring | odieresis |
| **155** | guilsinglright | cedilla | otilde |
| **156** | oe | *unused* | uacute |
| **157** | *unused* | hungarumlaut | ugrave |
| **158** | *unused* | ogonek | ucircumflex |

**Table 8: All Encodings Indexed By Byte Code**

| *Code* | *WindowsANSI* | *ISOLatin1* | *MacStandard* |
|---|---|---|---|
| **159** | Ydieresis | caron | udieresis |
| **160** | space | space | dagger |
| **161** | exclamdown | exclamdown | degree |
| **162** | cent | cent | cent |
| **163** | sterling | sterling | sterling |
| **164** | currency | currency | section |
| **165** | yen | yen | bullet |
| **166** | brokenbar | brokenbar | paragraph |
| **167** | section | section | germandbls |
| **168** | dieresis | dieresis | registered |
| **169** | copyright | copyright | copyright |
| **170** | ordfeminine | ordfeminine | trademark |
| **171** | guillemotleft | guillemotleft | acute |
| **172** | logicalnot | logicalnot | dieresis |
| **173** | hyphen | hyphen | notequal |
| **174** | registered | registered | AE |
| **175** | macron | macron | Oslash |
| **176** | degree | degree | infinity |
| **177** | plusminus | plusminus | plusminus |
| **178** | twosuperior | twosuperior | lessequal |
| **179** | threesuperior | threesuperior | greaterequal |
| **180** | acute | acute | yen |
| **181** | mu | mu | mu |
| **182** | paragraph | paragraph | partialdiff |
| **183** | periodcentered | periodcentered | summation |
| **184** | cedilla | cedilla | product |
| **185** | onesuperior | onesuperior | pi |
| **186** | ordmasculine | ordmasculine | integral |
| **187** | guillemotright | guillemotright | ordfeminine |
| **188** | onequarter | onequarter | ordmasculine |
| **189** | onehalf | onehalf | Omega |
| **190** | threequarters | threequarters | ae |
| **191** | questiondown | questiondown | oslash |
| **192** | Agrave | Agrave | questiondown |
| **193** | Aacute | Aacute | exclamdown |
| **194** | Acircumflex | Acircumflex | logicalnot |
| **195** | Atilde | Atilde | radical |
| **196** | Adieresis | Adieresis | florin |
| **197** | Aring | Aring | approxequal |
| **198** | AE | AE | Delta |
| **199** | Ccedilla | Ccedilla | guillemotleft |
| **200** | Egrave | Egrave | guillemotright |
| **201** | Eacute | Eacute | ellipsis |
| **202** | Ecircumflex | Ecircumflex | space |
| **203** | Edieresis | Edieresis | Agrave |
| **204** | Igrave | Igrave | Atilde |
| **205** | Iacute | Iacute | Otilde |
| **206** | Icircumflex | Icircumflex | OE |
| **207** | Idieresis | Idieresis | oe |
| **208** | Eth | Eth | endash |
| **209** | Ntilde | Ntilde | emdash |
| **210** | Ograve | Ograve | quotedblleft |

**Table 8: All Encodings Indexed By Byte Code**

| *Code* | *WindowsANSI* | *ISOLatin1* | *MacStandard* |
|---|---|---|---|
| **211** | Oacute | Oacute | quotedblright |
| **212** | Ocircumflex | Ocircumflex | quoteleft |
| **213** | Otilde | Otilde | quoteright |
| **214** | Odieresis | Odieresis | divide |
| **215** | multiply | multiply | lozenge |
| **216** | Oslash | Oslash | ydieresis |
| **217** | Ugrave | Ugrave | Ydieresis |
| **218** | Uacute | Uacute | fraction |
| **219** | Ucircumflex | Ucircumflex | currency |
| **220** | Udieresis | Udieresis | guilsinglleft |
| **221** | Yacute | Yacute | guilsinglright |
| **222** | Thorn | Thorn | fi |
| **223** | germandbls | germandbls | fl |
| **224** | agrave | agrave | daggerdbl |
| **225** | aacute | aacute | periodcentered |
| **226** | acircumflex | acircumflex | quotesinglbase |
| **227** | atilde | atilde | quotedblbase |
| **228** | adieresis | adieresis | perthousand |
| **229** | aring | aring | Acircumflex |
| **230** | ae | ae | Ecircumflex |
| **231** | ccedilla | ccedilla | Aacute |
| **232** | egrave | egrave | Edieresis |
| **233** | eacute | eacute | Egrave |
| **234** | ecircumflex | ecircumflex | Iacute |
| **235** | edieresis | edieresis | Icircumflex |
| **236** | igrave | igrave | Idieresis |
| **237** | iacute | iacute | Igrave |
| **238** | icircumflex | icircumflex | Oacute |
| **239** | idieresis | idieresis | Ocircumflex |
| **240** | eth | eth | apple |
| **241** | ntilde | ntilde | Ograve |
| **242** | ograve | ograve | Uacute |
| **243** | oacute | oacute | Ucircumflex |
| **244** | ocircumflex | ocircumflex | Ugrave |
| **245** | otilde | otilde | dotlessi |
| **246** | odieresis | odieresis | circumflex |
| **247** | divide | divide | tilde |
| **248** | oslash | oslash | macron |
| **249** | ugrave | ugrave | breve |
| **250** | uacute | uacute | dotaccent |
| **251** | ucircumflex | ucircumflex | ring |
| **252** | udieresis | udieresis | cedilla |
| **253** | yacute | yacute | hungarumlaut |
| **254** | thorn | thorn | ogonek |
| **255** | ydieresis | ydieresis | caron |

## C.2 Conversions from WindowsANSI Encoding

In Table 8, the first two columns give the byte code and name of a character in the source encoding vector, WindowsANSI (abbreviated for space in the table as ANSI). The third and fourth columns give the corresponding byte code in the destination encoding vectors, MacStandard (abbreviated as Mac) and ISOLatin1, respectively. The word “same” in a column means that the destination byte code is the same as the source byte code. The string “n/a” in a column means that the character has no equivalent in the destination encoding vector.

**Table 9: Conversions from WindowsANSI Encoding**

| *Character Name* | ***ANSI*** | *Mac* | *ISOLatin1* |
|---|---|---|---|
| quotesingle | **39** | *same* | *n/a* |
| hyphen | **45** | *same* | 173 |
| grave | **96** | *same* | 145 |
| quotesinglbase | **130** | 226 | *n/a* |
| florin | **131** | 196 | *n/a* |
| quotedblbase | **132** | 227 | *n/a* |
| ellipsis | **133** | 201 | *n/a* |
| dagger | **134** | 160 | *n/a* |
| daggerdbl | **135** | 224 | *n/a* |
| circumflex | **136** | 246 | 147 |
| perthousand | **137** | 228 | *n/a* |
| Scaron | **138** | *n/a* | *n/a* |
| guilsinglleft | **139** | 220 | *n/a* |
| OE | **140** | 206 | *n/a* |
| quoteleft | **145** | 212 | 96 |
| quoteright | **146** | 213 | 39 |
| quotedblleft | **147** | 210 | *n/a* |
| quotedblright | **148** | 211 | *n/a* |
| bullet | **149** | 165 | *n/a* |
| endash | **150** | 208 | *n/a* |
| emdash | **151** | 209 | *n/a* |
| tilde | **152** | 247 | 148 |
| trademark | **153** | 170 | *n/a* |
| scaron | **154** | *n/a* | *n/a* |
| guilsinglright | **155** | 221 | *n/a* |
| oe | **156** | 207 | *n/a* |
| bullet | **157** | 165 | *n/a* |
| bullet | **158** | 165 | *n/a* |
| Ydieresis | **159** | 217 | *n/a* |
| space | **160** | 32 | *same* |
| exclamdown | **161** | 193 | *same* |
| currency | **164** | 219 | *same* |
| yen | **165** | 180 | *same* |
| brokenbar | **166** | *n/a* | *same* |
| section | **167** | 164 | *same* |
| dieresis | **168** | 172 | *same* |
| ordfeminine | **170** | 187 | *same* |

**Table 9: Conversions from WindowsANSI Encoding (Continued)**

| *Character Name* | ***ANSI*** | *Mac* | *ISOLatin1* |
|---|---|---|---|
| guillemotleft | **171** | 199 | *same* |
| logicalnot | **172** | 194 | *same* |
| hyphen | **173** | 45 | *same* |
| registered | **174** | 168 | *same* |
| macron | **175** | 248 | *same* |
| degree | **176** | 161 | *same* |
| twosuperior | **178** | *n/a* | *same* |
| threesuperior | **179** | *n/a* | *same* |
| acute | **180** | 171 | *same* |
| paragraph | **182** | 166 | *same* |
| periodcentered | **183** | 225 | *same* |
| cedilla | **184** | 252 | *same* |
| onesuperior | **185** | *n/a* | *same* |
| ordmasculine | **186** | 188 | *same* |
| guillemotright | **187** | 200 | *same* |
| onequarter | **188** | *n/a* | *same* |
| onehalf | **189** | *n/a* | *same* |
| threequarters | **190** | *n/a* | *same* |
| questiondown | **191** | 192 | *same* |
| Agrave | **192** | 203 | *same* |
| Aacute | **193** | 231 | *same* |
| Acircumflex | **194** | 229 | *same* |
| Atilde | **195** | 204 | *same* |
| Adieresis | **196** | 128 | *same* |
| Aring | **197** | 129 | *same* |
| AE | **198** | 174 | *same* |
| Ccedilla | **199** | 130 | *same* |
| Egrave | **200** | 233 | *same* |
| Eacute | **201** | 131 | *same* |
| Ecircumflex | **202** | 230 | *same* |
| Edieresis | **203** | 232 | *same* |
| Igrave | **204** | 237 | *same* |
| Iacute | **205** | 234 | *same* |
| Icircumflex | **206** | 235 | *same* |
| Idieresis | **207** | 236 | *same* |
| Eth | **208** | *n/a* | *same* |
| Ntilde | **209** | 132 | *same* |

**Table 9: Conversions from WindowsANSI Encoding (Continued)**

| *Character Name* | ***ANSI*** | *Mac* | *ISOLatin1* |
|---|---|---|---|
| Ograve | **210** | 241 | *same* |
| Oacute | **211** | 238 | *same* |
| Ocircumflex | **212** | 239 | *same* |
| Otilde | **213** | 205 | *same* |
| Odieresis | **214** | 133 | *same* |
| multiply | **215** | *n/a* | *same* |
| Oslash | **216** | 175 | *same* |
| Ugrave | **217** | 244 | *same* |
| Uacute | **218** | 242 | *same* |
| Ucircumflex | **219** | 243 | *same* |
| Udieresis | **220** | 134 | *same* |
| Yacute | **221** | *n/a* | *same* |
| Thorn | **222** | *n/a* | *same* |
| germandbls | **223** | 167 | *same* |
| agrave | **224** | 136 | *same* |
| aacute | **225** | 135 | *same* |
| acircumflex | **226** | 137 | *same* |
| atilde | **227** | 139 | *same* |
| adieresis | **228** | 138 | *same* |
| aring | **229** | 140 | *same* |
| ae | **230** | 190 | *same* |
| ccedilla | **231** | 141 | *same* |
| egrave | **232** | 143 | *same* |
| eacute | **233** | 142 | *same* |
| ecircumflex | **234** | 144 | *same* |
| edieresis | **235** | 145 | *same* |
| igrave | **236** | 147 | *same* |
| iacute | **237** | 146 | *same* |
| icircumflex | **238** | 148 | *same* |
| idieresis | **239** | 149 | *same* |
| eth | **240** | *n/a* | *same* |
| ntilde | **241** | 150 | *same* |
| ograve | **242** | 152 | *same* |
| oacute | **243** | 151 | *same* |
| ocircumflex | **244** | 153 | *same* |
| otilde | **245** | 155 | *same* |
| odieresis | **246** | 154 | *same* |
| divide | **247** | 214 | *same* |
| oslash | **248** | 191 | *same* |
| ugrave | **249** | 157 | *same* |
| uacute | **250** | 156 | *same* |
| ucircumflex | **251** | 158 | *same* |
| udieresis | **252** | 159 | *same* |
| yacute | **253** | *n/a* | *same* |
| thorn | **254** | *n/a* | *same* |
| ydieresis | **255** | 216 | *same* |

## C.3 Conversions from MacStandard Encoding

In Table 9, the first two columns give the byte code and name of a character in the source encoding vector, MacStandard (abbreviated for space in the table as Mac). The third and fourth columns give the corresponding byte code in the destination encoding vectors, WindowsANSI (abbreviated as ANSI) and ISOLatin1, respectively. The word “same” in a column means that the destination byte code is the same as the source byte code. The string “n/a” in a column means that the character has no equivalent in the destination encoding vector.

**Table 10: Conversions from MacStandard Encoding**

| *Character Name* | ***Mac*** | *ANSI* | *ISOLatin1* |
|---|---|---|---|
| hyphen | **45** | *same* | 173 |
| grave | **96** | *same* | 145 |
| Adieresis | **128** | 196 | 196 |
| Aring | **129** | 197 | 197 |
| Ccedilla | **130** | 199 | 199 |
| Eacute | **131** | 201 | 201 |
| Ntilde | **132** | 209 | 209 |
| Odieresis | **133** | 214 | 214 |
| Udieresis | **134** | 220 | 220 |
| aacute | **135** | 225 | 225 |
| agrave | **136** | 224 | 224 |
| acircumflex | **137** | 226 | 226 |
| adieresis | **138** | 228 | 228 |
| atilde | **139** | 227 | 227 |
| aring | **140** | 229 | 229 |
| ccedilla | **141** | 231 | 231 |
| eacute | **142** | 233 | 233 |
| egrave | **143** | 232 | 232 |
| ecircumflex | **144** | 234 | 234 |
| edieresis | **145** | 235 | 235 |
| iacute | **146** | 237 | 237 |
| igrave | **147** | 236 | 236 |
| icircumflex | **148** | 238 | 238 |
| idieresis | **149** | 239 | 239 |
| ntilde | **150** | 241 | 241 |
| oacute | **151** | 243 | 243 |
| ograve | **152** | 242 | 242 |
| ocircumflex | **153** | 244 | 244 |
| odieresis | **154** | 246 | 246 |
| otilde | **155** | 245 | 245 |
| uacute | **156** | 250 | 250 |
| ugrave | **157** | 249 | 249 |
| ucircumflex | **158** | 251 | 251 |
| udieresis | **159** | 252 | 252 |
| dagger | **160** | 134 | *n/a* |
| degree | **161** | 176 | 176 |

**Table 10: Conversions from MacStandard Encoding (Continued)**

| *Character Name* | ***Mac*** | *ANSI* | *ISOLatin1* |
|---|---|---|---|
| section | **164** | 167 | 167 |
| bullet | **165** | 149 | *n/a* |
| paragraph | **166** | 182 | 182 |
| germandbls | **167** | 223 | 223 |
| registered | **168** | 174 | 174 |
| trademark | **170** | 153 | *n/a* |
| acute | **171** | 180 | 146 |
| dieresis | **172** | 168 | 152 |
| notequal | **173** | *n/a* | *n/a* |
| AE | **174** | 198 | 198 |
| Oslash | **175** | 216 | 216 |
| infinity | **176** | *n/a* | *n/a* |
| lessequal | **178** | *n/a* | *n/a* |
| greaterequal | **179** | *n/a* | *n/a* |
| yen | **180** | 165 | 165 |
| partialdiff | **182** | *n/a* | *n/a* |
| summation | **183** | *n/a* | *n/a* |
| product | **184** | *n/a* | *n/a* |
| pi | **185** | *n/a* | *n/a* |
| integral | **186** | *n/a* | *n/a* |
| ordfeminine | **187** | 170 | 170 |
| ordmasculine | **188** | 186 | 186 |
| Omega | **189** | *n/a* | *n/a* |
| ae | **190** | 230 | 230 |
| oslash | **191** | 248 | 248 |
| questiondown | **192** | 191 | 191 |
| exclamdown | **193** | 161 | 161 |
| logicalnot | **194** | 172 | 172 |
| radical | **195** | *n/a* | *n/a* |
| florin | **196** | 131 | *n/a* |
| approxequal | **197** | *n/a* | *n/a* |
| Delta | **198** | *n/a* | *n/a* |
| guillemotleft | **199** | 171 | 171 |
| guillemotright | **200** | 187 | 187 |
| ellipsis | **201** | 133 | *n/a* |
| space | **202** | 32 | 32 |

**Table 10: Conversions from MacStandard Encoding (Continued)**

| *Character Name* | ***Mac*** | *ANSI* | *ISOLatin1* |
|---|---|---|---|
| Agrave | **203** | 192 | 192 |
| Atilde | **204** | 195 | 195 |
| Otilde | **205** | 213 | 213 |
| OE | **206** | 140 | *n/a* |
| oe | **207** | 156 | *n/a* |
| endash | **208** | 150 | *n/a* |
| emdash | **209** | 151 | *n/a* |
| quotedblleft | **210** | 147 | *n/a* |
| quotedblright | **211** | 148 | *n/a* |
| quoteleft | **212** | 145 | 96 |
| quoteright | **213** | 146 | 39 |
| divide | **214** | 247 | 247 |
| lozenge | **215** | *n/a* | *n/a* |
| ydieresis | **216** | 255 | 255 |
| Ydieresis | **217** | 159 | *n/a* |
| fraction | **218** | *n/a* | *n/a* |
| currency | **219** | 164 | 164 |
| guilsinglleft | **220** | 139 | *n/a* |
| guilsinglright | **221** | 155 | *n/a* |
| fi | **222** | *n/a* | *n/a* |
| fl | **223** | *n/a* | *n/a* |
| daggerdbl | **224** | 135 | *n/a* |
| periodcentered | **225** | 183 | 183 |
| quotesinglbase | **226** | 130 | *n/a* |
| quotedblbase | **227** | 132 | *n/a* |
| perthousand | **228** | 137 | *n/a* |
| Acircumflex | **229** | 194 | 194 |
| Ecircumflex | **230** | 202 | 202 |
| Aacute | **231** | 193 | 193 |
| Edieresis | **232** | 203 | 203 |
| Egrave | **233** | 200 | 200 |
| Iacute | **234** | 205 | 205 |
| Icircumflex | **235** | 206 | 206 |
| Idieresis | **236** | 207 | 207 |
| Igrave | **237** | 204 | 204 |
| Oacute | **238** | 211 | 211 |
| Ocircumflex | **239** | 212 | 212 |
| apple | **240** | *n/a* | *n/a* |
| Ograve | **241** | 210 | 210 |
| Uacute | **242** | 218 | 218 |
| Ucircumflex | **243** | 219 | 219 |
| Ugrave | **244** | 217 | 217 |
| dotlessi | **245** | *n/a* | 144 |
| circumflex | **246** | *n/a* | 147 |
| tilde | **247** | 152 | 148 |

**Table 10: Conversions from MacStandard Encoding (Continued)**

| *Character Name* | ***Mac*** | *ANSI* | *ISOLatin1* |
|---|---|---|---|
| macron | **248** | 175 | 149 |
| breve | **249** | *n/a* | 150 |
| dotaccent | **250** | *n/a* | 151 |
| ring | **251** | *n/a* | 154 |
| cedilla | **252** | 184 | 155 |
| hungarumlaut | **253** | *n/a* | 157 |
| ogonek | **254** | *n/a* | 158 |
| caron | **255** | *n/a* | 159 |

## C.4 Conversions from ISOLatin1 Encoding

In Table 10, the first two columns give the byte code and name of a character in the source encoding vector, ISOLatin1 . The third and fourth columns give the corresponding byte code in the destination encoding vectors, WindowsANSI (abbreviated for space in the table as ANSI) and MacStandard (abbreviated as Mac), respectively. The word “same” in a column means that the destination byte code is the same as the source byte code. The string “n/a” in a column means that the character has no equivalent in the destination encoding vector.

**Table 11: Conversions from ISOLatin1 Encoding**

| *Character Name* | ***ISOLatin1*** | *ANSI* | *Mac* |
|---|---|---|---|
| quoteright | **39** | 146 | 213 |
| quoteleft | **96** | 145 | 212 |
| dotlessi | **144** | *n/a* | 245 |
| grave | **145** | 96 | 96 |
| acute | **146** | 180 | 171 |
| circumflex | **147** | *n/a* | 246 |
| tilde | **148** | 152 | 247 |
| macron | **149** | 175 | 248 |
| breve | **150** | *n/a* | 249 |
| dotaccent | **151** | *n/a* | 250 |
| dieresis | **152** | 168 | 172 |
| ring | **154** | *n/a* | 251 |
| cedilla | **155** | *n/a* | 252 |
| hungarumlaut | **157** | *n/a* | 253 |
| ogonek | **158** | *n/a* | 254 |
| caron | **159** | *n/a* | 255 |
| space | **160** | *same* | 32 |
| exclamdown | **161** | *same* | 193 |
| currency | **164** | *same* | 219 |
| yen | **165** | *same* | 180 |
| brokenbar | **166** | *same* | *n/a* |
| section | **167** | *same* | 164 |
| dieresis | **168** | *same* | 172 |
| ordfeminine | **170** | *same* | 187 |
| guillemotleft | **171** | *same* | 199 |
| logicalnot | **172** | *same* | 194 |
| hyphen | **173** | *same* | 45 |
| registered | **174** | *same* | 168 |
| macron | **175** | *same* | 248 |
| degree | **176** | *same* | 161 |
| twosuperior | **178** | *same* | *n/a* |
| threesuperior | **179** | *same* | *n/a* |
| acute | **180** | *same* | 171 |
| paragraph | **182** | *same* | 166 |
| periodcentered | **183** | *same* | 225 |

**Table 11: Conversions from ISOLatin1 Encoding (Continued)**

| *Character Name* | ***ISOLatin1*** | *ANSI* | *Mac* |
|---|---|---|---|
| cedilla | **184** | *same* | 252 |
| onesuperior | **185** | *same* | *n/a* |
| ordmasculine | **186** | *same* | 188 |
| guillemotright | **187** | *same* | 200 |
| onequarter | **188** | *same* | *n/a* |
| onehalf | **189** | *same* | *n/a* |
| threequarters | **190** | *same* | *n/a* |
| questiondown | **191** | *same* | 192 |
| Agrave | **192** | *same* | 203 |
| Aacute | **193** | *same* | 231 |
| Acircumflex | **194** | *same* | 229 |
| Atilde | **195** | *same* | 204 |
| Adieresis | **196** | *same* | 128 |
| Aring | **197** | *same* | 129 |
| AE | **198** | *same* | 174 |
| Ccedilla | **199** | *same* | 130 |
| Egrave | **200** | *same* | 233 |
| Eacute | **201** | *same* | 131 |
| Ecircumflex | **202** | *same* | 230 |
| Edieresis | **203** | *same* | 232 |
| Igrave | **204** | *same* | 237 |
| Iacute | **205** | *same* | 234 |
| Icircumflex | **206** | *same* | 235 |
| Idieresis | **207** | *same* | 236 |
| Eth | **208** | *same* | *n/a* |
| Ntilde | **209** | *same* | 132 |
| Ograve | **210** | *same* | 241 |
| Oacute | **211** | *same* | 238 |
| Ocircumflex | **212** | *same* | 239 |
| Otilde | **213** | *same* | 205 |
| Odieresis | **214** | *same* | 133 |
| multiply | **215** | *same* | *n/a* |
| Oslash | **216** | *same* | 175 |
| Ugrave | **217** | *same* | 244 |
| Uacute | **218** | *same* | 242 |

**Table 11: Conversions from ISOLatin1 Encoding (Continued)**

| *Character Name* | ***ISOLatin1*** | *ANSI* | *Mac* |
|---|---|---|---|
| Ucircumflex | **219** | *same* | 243 |
| Udieresis | **220** | *same* | 134 |
| Yacute | **221** | *same* | *n/a* |
| Thorn | **222** | *same* | *n/a* |
| germandbls | **223** | *same* | 167 |
| agrave | **224** | *same* | 136 |
| aacute | **225** | *same* | 135 |
| acircumflex | **226** | *same* | 137 |
| atilde | **227** | *same* | 139 |
| adieresis | **228** | *same* | 138 |
| aring | **229** | *same* | 140 |
| ae | **230** | *same* | 190 |
| ccedilla | **231** | *same* | 141 |
| egrave | **232** | *same* | 143 |
| eacute | **233** | *same* | 142 |
| ecircumflex | **234** | *same* | 144 |
| edieresis | **235** | *same* | 145 |
| igrave | **236** | *same* | 147 |
| iacute | **237** | *same* | 146 |
| icircumflex | **238** | *same* | 148 |
| idieresis | **239** | *same* | 149 |
| eth | **240** | *same* | *n/a* |
| ntilde | **241** | *same* | 150 |
| ograve | **242** | *same* | 152 |
| oacute | **243** | *same* | 151 |
| ocircumflex | **244** | *same* | 153 |
| otilde | **245** | *same* | 155 |
| odieresis | **246** | *same* | 154 |
| divide | **247** | *same* | 214 |
| oslash | **248** | *same* | 191 |
| ugrave | **249** | *same* | 157 |
| uacute | **250** | *same* | 156 |
| ucircumflex | **251** | *same* | 158 |
| udieresis | **252** | *same* | 159 |
| yacute | **253** | *same* | *n/a* |
| thorn | **254** | *same* | *n/a* |
| ydieresis | **255** | *same* | 216 |

# Appendix D: Changes Since Earlier Versions

## D.1 Changes since Version 4.0, October 14, 1992

- Changed spec version number from 4.0 to 4.1 in all appropriate places.
- Changed this section from Appendix C to Appendix D.
- Inserted new Appendix C, Character Encodings, for use with the new *LanguageEncoding keyword.
- Added the following new keywords

| | | |
|---|---|---|
| *TTRasterizer | *LanguageEncoding | *ShortNickName |
| *ColorModel | *?ColorModel | *DefaultColorModel |
| *JCLOpenUI | *JCLCloseUI | *JCLToPSInterpreter |
| *JCLBegin | *JCLEnd | |
| *JCLFrameBufferSize | *?JCLFrameBufferSize | *DefaultJCLFrameBufferSize |
| *JCLResolution | *?JCLResolution | *DefaultJCLResolution |
| *MaxMediaHeight | *?CurrentMediaHeight | |

- Substantially rewrote section 5.12, *Custom Page Sizes*, to define the meaning of custom page sizes on cut-sheet devices (old version dealt only with roll-fed devices). Added definitions for cut-sheet devices to the custom page size parameters and to all relevant keywords. Divided roll-fed and cut-sheet devices into two subsections, wrote new intro to cover both sections. Clarified portions of *HWMargins and added info about how to use it. Added new illustrations and examples. Changed *MaxMediaWidth from *int* to *real.* Added explanation to *CurrentMediaWidth.
- Added new section 5.23, “Features Accessible Only Through Job Control Language”, to document new *JCL keywords.
- In section 3.6, “Syntax of Values”, under the subheadings *QuotedValues* and *Parsing Summary For Values*, added exception for *JCL keywords to the first rule, regarding the presence of option keywords. *JCL keywords are treated like QuotedValues even if they have an option and look like InvocationValues.

- In section 4.2, *Elementary Types*, added new elementary type: JCL. Under *Protocols, added note to subsection on PJL regarding the interaction of the PJL value and the *JCL keywords. Added JCLSetup section to *OrderDependency.

- Added reference to *ShortNickName in *NickName description. Also under *NickName, clarified use of translation strings and encodings with *NickName.

- Changed description of *UIConstraints and the Installable Options section to include keyword-value pairs as well as keyword-option pairs.

- *ImageableArea: Added description of PPD files for devices that have pages with an imageable area that can vary depending on resolution and other factors.

- Added subheading *Syntax and Use* in section 5.7, *Installable Options*. Added new section of info: *Keyword-Value Pairs*.

- Various minor wording changes were made for clarification or brevity. Minor typographical errors were fixed. Updated examples at end to include some of the new keywords. Pages renumbered, new index and TOC generated.

## D.2 Changes since February 14, 1992

- Minor typographical errors were corrected.

## D.3 Changes since Version 3.0, dated March 8, 1989

### D.3.1 Changes to Text

Significant rewriting and reorganizing occurred in this version of the spec, so rather than documenting line-by-line changes, only the major semantic and syntactical changes are described here.

- The specification version number was increased to 4.0

- International headquarters' addresses added to front cover; updated copyright.

- "PostScript Printer Description files" was changed to "PPD files" in all but the first few times it is mentioned; "Printer Description files" were likewise changed to "PPD files" in all cases. Changed "printer" to "device" in most cases. Changed "paper" to "media" in text, not in keywords.

- The option keyword section at the end of the document was removed; all currently registered option keywords are now documented with their respective main keywords.

- Added section with several sample PPD files.

- *Introduction* became a section (section 1) and was rewritten to reflect new focus on building a user interface from a PPD file and to get more basic information on the first page.

- *Using PPD Files*: Completely rewritten to show how document composition application and print manager interact to create PostScript language code, and how code sample grows as it passes through various phases (DSC comments added). Added sections on building a user interface, inserting print-time features, error-handling, post-processing, and order dependencies within a file.

- *Local Customization and *Include:* The entire section was rewritten to explain what kind of information is in the initial PPD file, what kind of information a user or system administrator might want to change or add, the drawbacks of editing a PPD file directly, and alternative suggestions to managing PPD files. Explained local customization files in more detail and emphasized consistent use of that title for them, since we never had a title before. Added subsection on changing *Default- values in local customization file. Expanded meaning of defaults—in original PPD file, defaults are the factory defaults, but they can now be changed in a local customization file. New rule for *Include: filenames must be enclosed in quotes.

- *The Format*: Significantly rewritten. Added sentence about how queries only work if the physical interface to the device allows feedback. Added ASCII code chart for commonly referenced characters, definition of terms, and descriptions of canonical forms of keyword entries. A bullet was added to point out the maximum line length of 255 characters. The maximum length of 40 characters per keyword was clarified under *Main Keywords*. Split apart *Parsing Details* and reintegrated into subsections on main keywords, option keywords, and values. Divided descriptions of main keywords, option keywords, and values into subsections for PPD writers and PPD parsers.

- *Details*: Section was split apart and integrated into separate sections on main keywords, option keywords, and values. Under *Main Keywords*, it was clarified that a *grep* for a complete keyword includes the *asterisk* in the keyword name (so "*PageSize" is not a substring of "*DefaultPageSize").

- *Semantics of Main Keywords*: Section was removed and information moved to either *The Format* or *Main Keywords*.

- The following keywords are now *required* in a a PPD file, whereas previously there was no requirement. Some are old keywords, some are new: *PPD-Adobe, *Product, *PSVersion, *PCFileName, *ModelName, *NickName, *PageSize, *PageRegion, *ImageableArea, *PaperDimension, *FileVersion, *FormatVersion, and *LanguageVersion.

- *Option Keywords*: Significantly rewritten. Added advice for parsers and emphasized extensibility of option keywords.

- *Translation String Syntax*: This section was moved and retitled from *Foreign Language Customization: Translation String Syntax*, because it applies to more than foreign languages. Section was expanded to include examples of translating cryptic keywords "from English to English". In the French example, the nonexistent keyword *PaperSize was changed to *PageSize and syntactically incorrect percent signs and brackets and the word PrinterError were all removed. Added section about 7-bit ASCII PPD files and how to represent 8-bit characters (for foreign languages) as hex strings. Provided reasons and noted that translation strings, if present, should always be displayed to the user rather than the original option keyword. Added *Parsing Summary* for translation strings.

- *Human-Readable Comments*: Added paragraph about comments in PostScript language code.

- *PostScript Language Sequences*: The prohibition against leaving anything on the operand and dictionary stacks was removed, as it is already violated by the color separation keywords, the halftone screen keywords, the transfer function keywords, and probably others.

- *Parsing Details* section was removed and integrated into previous sections.

- *Syntax of Specification*: New section to document syntax of spec itself. Added syntax and elementary types. Changed the symbols used for "or" in the meta-syntax from a *slash* to a *vertical bar,* to be consistent with the DSC. Inclusive "or" is now defined to be ellipsis, like the DSC. Added explanations and examples of each type of PPD entry (main keyword with fixed option list, main keyword with variable option list, and keyword with no options).

- *Paper Handling*, was merged with a later section, *Introduction to Media Handling*.

- The *Color Extensions* section was removed and its material was moved to the beginning of the *Color Keywords* section.

- In the *Keywords* intro, added paragraph about how if a feature is not supported by a device, it should be omitted from the PPD. Moved *Standard Option Values For Main Keywords* from back of document to beginning of

section, to document global options like True, False, None, and Unknown. Added examples for each of these and added note about not using None or Unknown to indicate absence of a feature on a device.

UI • *Keywords*: Rearranged all keywords into more logical sections and order. Removed all option keywords from end of document and integrated them into their respective main keyword sections. Added UI symbol (seen to left of this paragraph) throughout document to mark keywords that should be bracketed with *OpenUI/*CloseUI.

### D.3.2 New Keywords

| | | |
|---|---|---|
| *AdvanceMedia | *?AdvanceMedia | *DefaultAdvanceMedia |
| *BindColor | *?BindColor | *DefaultBindColor |
| *BindEdge | *?BindEdge | *DefaultBindEdge |
| *BindType | *?BindType | *DefaultBindType |
| *BindWhen | *?BindWhen | *DefaultBindWhen |
| *BitsPerPixel | *?BitsPerPixel | *DefaultBitsPerPixel |
| *BlackSubstitution | *?BlackSubstitution | *DefaultBlackSubstitution |
| *Booklet | *?Booklet | *DefaultBooklet |
| *Collate | *?Collate | *DefaultCollate |
| *CutMedia | *?CutMedia | *DefaultCutMedia |
| *Duplex | *?Duplex | *DefaultDuplex |
| *FoldType | *?FoldType | *DefaultFoldType |
| *FoldWhen | *?FoldWhen | *DefaultFoldWhen |
| *InsertSheet | *?InsertSheet | *DefaultInsertSheet |
| *Jog | *?Jog | *DefaultJog |
| *MediaColor | *?MediaColor | *DefaultMediaColor |
| *MediaType | *?MediaType | *DefaultMediaType |
| *MediaWeight | *?MediaWeight | *DefaultMediaWeight |
| *MirrorPrint | *?MirrorPrint | *DefaultMirrorPrint |
| *NegativePrint | *?NegativePrint | *DefaultNegativePrint |
| *OutputMode | *?OutputMode | *DefaultOutputMode |
| *Separations | *?Separations | *DefaultSeparations |
| *Signature | *?Signature | *DefaultSignature |
| *Slipsheet | *?Slipsheet | *DefaultSlipsheet |
| *Smoothing | *?Smoothing | *DefaultSmoothing |
| *Sorter | *?Sorter | *DefaultSorter |
| *StapleLocation | *?StapleLocation | *DefaultStapleLocation |
| *StapleOrientation | *?StapleOrientation | *DefaultStapleOrientation |
| *StapleWhen | *?StapleWhen | *DefaultStapleWhen |
| *StapleX | *?StapleX | *DefaultStapleX |
| *StapleY | *?StapleY | *DefaultStapleY |
| *TraySwitch | *?TraySwitch | *DefaultTraySwitch |

| | | |
|---|---|---|
| *OpenUI | *CloseUI | *Extensions |
| *OpenGroup | *CloseGroup | *Protocols |
| *StartEmulator_ | *StopEmulator_ | *Emulators |
| *FaxSupport | *JobPatchFile | *?PatchFile |
| *CustomPageSize | *ParamCustomPageSize | *?CurrentMediaWidth |

| | | |
|---|---|---|
| *MaxMediaWidth | *CenterRegistered | *PageStackOrder |
| *Resolution | *HWMargins | *LandscapeOrientation |
| *ColorRenderDict | *OrderDependency | *PCFileName |
| *DefaultColorSpace | *LanguageLevel | *ModelName |
| *RequiresPageRegion | *UIConstraints | *AccurateScreensSupport |

### D.3.3 Changes to Existing Keywords

The changes to the syntax and semantics of actual keywords from version 3.0 are as follows:

- *Include: The filename must now be enclosed in double quotes.
- *ImageableArea, *?ImageableArea: The numbers in the value (and the numbers returned by the query) are now real numbers; previously, they were integers.
- *DefaultResolution, *?Resolution, *SetResolution: These can now take an option of the form 300x600dpi. Previously, the only format for an option was 300dpi. This change is necessary to accommodate printers with anamorphic resolution. See the descriptions of these keywords for full details.
- *Font: Two more fields were added to the value to describe the *character set* of the font and whether the font is *removable* or not. The value now has four fields instead of the previous two fields. The two new fields were added at the end of the old value, after the version number.
- *PaperTray, *?PaperTray, and *DefaultPaperTray were removed, as their code had always been redundant with *PageSize and no tools were found to depend upon their presence.
- *Collator, *?Collator, *DefaultCollator: These were changed to *Collate, *?Collate, and *DefaultCollate, since they had not previously been used in PPD files and this brought them more in line with other keyword usage.

### D.3.4 Changes to Descriptions of Existing Keywords

- *General Information Keywords*: This section title was changed from *General Defaults and Information Keywords*. In *FileVersion, the structure of the version number and how to update it was clarified. In *FormatVersion, the conformance number of the spec was changed to “4.0.” In *LanguageVersion, added material about the encoding of foreign translation strings and how to represent non-English characters in translation strings. Added new language option keywords Swedish and Danish. Added Level 1 and Level 2 code fragments to *Product. The definition and examples of *Nickname were corrected to be a string within quotation

marks but without parentheses (for example, “Apple LaserWriter® II NTX v49.3”), since all existing PPD files had been built that way (without parentheses).

- *Basic Device Capabilities*: *ColorDevice: clarified that this keyword indicates physical color output. Added reference to *Extensions, for devices that support color extensions but may or may not physically output color. Moved *FileSystem keywords here. *FileSystem was clarified as referring to the *capacity* for a file system; example was added for a device that has the *capacity* but does not have a file system *installed*; reemphasized that this entry should be omitted if there is no capacity for having a file system. Clarified meaning of return values of *?FileSystem. Moved *Throughput to this section.

- *Keywords*: Added new section *Structure Keywords*. Moved *Include and *End to this section.

- *Introduction to Media Handling*: New section created from *Paper Handling*. Significantly rewritten. Added **setpagedevice** to the list of example invocations.

- *Media Option Keywords*: New section, created from several old sections. Significantly rewritten. Added info about the extensibility of media option keywords. *PaperKeyword* became *mediaOption* throughout the document. Refined explanation of how to handle Envelopes. Prose was added about parsing for *OpenUI/*CloseUI rather than a specific list of options. Clarified meaning of Transverse (long edge perpendicular to feed direction).

- *Media Option Keywords*: Rewrote all dimensions in consistent format. Built tables of ISO and JIS standard paper and envelope sizes and added most sizes. Moved most U.S. standard definitions to a separate table, except for the ones that needed extra text to explain them, and included several new media sizes. Changed imageable area definition of A4Small from inches to points (all others were already in points). Changed imageable area of LetterSmall from 553x731.5 points to 552x730 points, because that is what 8 out of 9 PPD files had for that imageable area.

- *Paper Size Invocation* became *Media Selection* Under *DefaultPageSize, Unknown is now an option. Clarified, with examples, how *PageSize is meant to be used. Explained how *PageRegion should be used. Removed *PaperTray and associated default and query.

- *Information About Media Sizes*: Under *DefaultImageableArea and *DefaultPaperDimension, the sentence “The value should always be Letter.” was removed. “This value may be Unknown or one of the media options listed under *ImageableArea/*PaperDimension.” was added. For *ImageableArea, clarified that imageable area was measured in PostScript

default units. Changed “integers” to “reals” for all imageable area and paper dimension keywords. Added that x and y axes should correspond in *ImageableArea and *PaperDimension.

- *Media Handling Features*: For the *InputSlot keywords, the list of options was replaced by *trayOption* and explanation was added. Current options were brought forward from the rear of the document. In the definition of *ManualFeed the value None was removed from the list of valid choices and the explanation changed. The *OutputBin keywords were changed to accept an extensible list of bin names, the current options were integrated into this section from the rear of the document, and return values were specified for the query. Added Rear option for output trays. The explanation of *OutputOrder was modified to address how most devices handle page stack order today.

- *Keywords*: Moved information on device resolution to new section *Resolution and Appearance Control*

- *Resolution and Appearance Control*: This new section was created to contain *SetResolution, *DefaultResolution, *?Resolution, and resolution information. Expanded format of the *resolution* option keyword to include “300x300dpi” (as well as the old format of “300dpi”), to accommodate devices with anamorphic resolution.

- *Gray Levels & Halftoning*: The description on *ScreenFreq was changed from “the second argument” to “the *frequency* argument”. The description on *ScreenAngle was changed from “the first argument” to “the *angle* argument”. Both *ScreenFreq and *ScreenAngle were changed to be a *real* instead of an *integer*, since that is what is returned by **currentscreen**. Throughout this section, “the .Invert qualifier” was changed to “the .Inverse qualifier”. This was a typo in the spec; the .Invert qualifier never appeared in a PPD file. Spot options were integrated into this section.

- *Gray Levels & Halftoning*: Several option keywords were added to *Transfer to include the ability to define transfer functions for each process color. Options added to Null and Normalized were Red, Green, and Blue. The option Factory was added to distinguish between transfer functions that are built-in and transfer functions that are suggested. Entire section was rewritten for more detail and clarification.

- *Color Separation Keywords*: Merged earlier color separation section (used to be 3.0) into the intro of this section. Added DiamondDot spot function.

- *Font Related*: In the definition of *DefaultFont, Error was included as a possible value.

- *Font Related*: *Font was expanded to include a charset field to describe the character set supported by the font. A new value field, status, with possible values of ROM or Disk, was added to describe whether or not fonts are removable. Built table to show the difference between ROM and Disk. Integrated font option keywords for encoding and charset into this section.

- *Printer Messages*: General rewording throughout section. Under *Message, added “Messages that appear under *Status or *PrinterError should not be repeated here.” Clarified that same messages may appear both under *Status and *PrinterError. Added examples of PPD file entries and translation strings to *PrinterError and added Level 2 device names to the list of options for *Source.

- *System Management*: Reworded *PatchFile explanation for clarity and added requirements for behavior of patch file code. Reformatted *FreeVM to call out code. The description of *Password was changed to refer to the *current* password instead of the *default* password.

- *System Management*: Under *DeviceAdjustMatrix, added that this entry should be commented out if it is not used, and added reference to *Localization* section (for instructions about creating a local customization file for a device that needs to use *DeviceAdjustMatrix).

- Cleaned up all sample code to eliminate “begin...end” so no dictionaries are left on the stack if the code fails.

# Index

## Symbols



## A



## B



## C

## D



## E



## F, G

## H



## I



## J, K



## L



## M

## N



## O



## P



## Q



## R

## S



## T



## U



## V



## W, X, Y, Z