POSTSCRIPT®
Software From Adobe

PostScript Printer Description File Format Specification

Adobe Developer Support

Version 4.3

9 February 1996

Adobe Systems Incorporated

Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/

PN LPS5003

Contents

PostScript Printer Description File Format Specification

1 Introduction

PostScript™ printer description files (PPD files) are text files that provide a uniform approach to using the diverse features of devices that contain PostScript interpreters. Such 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. PPD files provide applications with the necessary information about a device's features, including the feature options, the default settings, how to request the current settings, how to change the settings, and other information that might be used for scheduling jobs.

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. Applications on the host computer can then parse PPD files to discover the list of features available on a 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 appropriate place 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 changes to the printer, such as the addition of fonts or memory, or to configure a device a certain way (for example, to always print in duplex).

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 that code into the output file before routing the file to the new device.

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.

1.1 ASCII Code Chart

The following ASCII characters are referenced repeatedly in this document:

- *asterisk*, ‘*’ (decimal 42)
- *caret*, ‘^’ (decimal 94)
- *colon*, ‘:’ (decimal 58)
- *double quote*, ‘"’ (decimal 34)
- *newline* — any combination of carriage return (decimal 13) and line feed (decimal 10)
- *period,* ‘.’ (decimal 46)
- *question mark*, ‘?’ (decimal 63)
- *slash*, ‘/’ (decimal 47)
- *space* (decimal 32)
- *tab* (horizontal) (decimal 9)
- *open angle bracket*, ‘<‘ (decimal 60)
- *closing angle bracket*, ‘>’ (decimal 62)

1.2 Definition of Terms

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

There are two basic types of keywords in a PPD file: *main keywords* and *option keywords*. Main keywords denote a device feature, such as the set of available page sizes (*PageSize) or input slots (*InputSlot).

Option keywords, which modify main keywords, describe the list of available options for a feature. For example, the option keywords for the main 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.

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. Default keywords share the root name of a main keyword, as in *DefaultPageSize and *PageSize. When discussed as a class of keywords, default keywords are also referred to as *Default keywords, because *Default is always their prefix. A *stand-alone* default keyword is one that appears in the PPD file without its related main keyword (for example, *DefaultResolution without *Resolution).

A *query keyword* provides a code sequence, which, when downloaded to the device, returns information about the current state of the device. This can be used by applications to determine the state of a device and perhaps request operator intervention (for example, if the appropriate media tray is not present). Note that queries can only be used when the physical interface to the device permits feedback from the device. Also, queries must be emitted in a "query job" that immediately precedes the print job. Among other things, this allows spoolers to process queries without processing the actual print job. (See the *DSC* specification for a description of query jobs.)

Not every main keyword has an associated query keyword. Query keywords have been defined only if they are possible and useful, and are completely optional.

A *statement* is a single instance of a main keyword, option keyword (if any), and value. There are seven formats for statements:

1. *MainKey

2. *MainKey: StringValue

3. *MainKey: "QuotedValue"

4. *MainKey: ^SymbolValue

5. *MainKey OptionKey: StringValue

6. *MainKey OptionKey: "InvocationValue"

7. *MainKey OptionKey: ^SymbolValue

Each statement in a PPD file falls into one of these formats. The value types are defined in section 3.4.

An *entry* describes a group of statements that logically belong together. An entry usually includes a *Default keyword, several instances of a main keyword with different option keywords and values, and a query keyword. An entry often also has surrounding *structure keywords*, which are discussed in section 5.2.

There are two general classes of main keywords: *informational* and *UI* (for User Interface). *Informational* main keywords provide information about a feature, such as how much memory is available or which fonts are resident. Such information is usually only useful to an application and does not appear in the user interface. *UI keywords* represent features that would commonly appear in a user interface (UI). They provide the code to invoke a user-selectable feature within the context of a print job; for example, the selection of an input tray or manual feed. The entries of UI keywords are surrounded by the structure keywords *OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI (see section 5.2).

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 the ASCII characters carriage return (decimal 13) and line feed (decimal 10). In this specification, the set of line termination characters is referred to as *newline*.

White space is defined as any combination of the ASCI 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 a statement (exceptions to this rule are noted on a case-by-case basis).

The following *8-bit byte codes* are allowed in a PPD file: decimal 32 through decimal 255 inclusive, plus decimal 9 (ASCII horizontal tab), decimal 10 (ASCII line feed), and decimal 13 (ASCII carriage return). However, most data types further restrict the allowable byte code range. Characters that fall within the allowable range for a particular data type are called *in-range byte codes*. Characters that fall outside the allowable range for a particular data type are called *out-of-range byte codes.*

Printable 7-bit ASCII is the set of byte codes that fall within the range of decimal 32 through decimal 126 inclusive, plus decimal 9 (ASCII horizontal tab), decimal 10 (ASCII line feed), and decimal 13 (ASCII carriage return).

A *hexadecimal substring* is used to represent out-of-range byte codes in certain data types (see section 3.5). 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 and tabs can be intermixed with the hexadecimal digits and should be ignored. Newlines may occur and should be ignored, except in translation strings, where they are illegal. See section 3.5 for the treatment of newlines in translation strings. All other characters should be considered an error. An odd number of hexadecimal digits is also an error.

2 Using PPD Files

PPD files can be used during several phases of document production. First, during printer installation or setup, 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. An application can then parse the PPD file for a list of 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 determine which options to display to the user at print time.

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 to fold paper. The PPD file will contain information for all features that 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, and to tell the printing application that they are available.

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. The default values for those features can be obtained from the PPD file. Where applicable, there is also code that can be used to determine the current settings of features (known as querying the device). 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. More detail on the use of PPD files in each phase of document production is provided in the next few sub-sections.

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.

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 and the paper will be drawn from a tray other than the manual feed tray. The PPD file can thus be used to tell users both what they can do and what will happen if they do nothing.

It is important to realize that the defaults in the PPD file do not necessarily reflect the current state of the device, as a system manager or a previous job could have changed the state of the device. It is also important to realize that a print manager is not required to use the PPD defaults as an initial starting point for display. Some print managers save the user's previous job settings and use those as initial settings, rather than using the device's default settings.

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 if 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. 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 the DSC comments and the device-specific code added.

```
%!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
%%EOF
```

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
   (<<) cvx exec /PageSize [612 792] (>>) cvx exec setpagedevice
%%EndFeature
%%EndSetup
%%Page: one 1
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
%%EOF
```

2.4 Error Handling

Print managers should include a reasonable level of error-handling, both when parsing PPD files and when downloading code from PPD files to a device. Examples of possible errors in a PPD file are dangling symbolic references (section 5.23) missing information about page sizes (section 5.12, section 5.14, and section 5.15), and missing required keywords (beginning of section 5).

When querying a device, solicited and unsolicited status messages from the device may interrupt the transmission of the query return value. On some communication channels, this may cause buffer overflow when the maximum number of characters retrievable by the host is exceeded. In such an environment, the print manager can alleviate the problem by limiting the number of queries sent at one time, waiting some period of time (such as 100 milliseconds) between queries, and by flushing the communication channel between queries.

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.

Here is an example of error-handling code, using the same sample document shown in previous examples:

```
%!PS-Adobe-3.0
%%Title: test.ps
%%LanguageLevel: 2
%%EndComments
/sp /showpage load def
%%EndProlog
%%BeginSetup
countdictstack[{
%%BeginFeature: *PageSize Letter
(<<) cvx exec /PageSize [612 792] (>>) cvx exec setpagedevice
%%EndFeature
}stopped
cleartomark
countdictstack exch sub dup 0 gt
{
     { end } repeat
}{
     pop
}ifelse
%%EndSetup
%%Page: one 1
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
%%EOF
```

In this example, it is important that this line of code

```
countdictstack[{
```

appears **before** the %%BeginFeature: comment line, and that this line

```
}stopped
```

and the lines following it appear **after** the %%EndFeature line. Otherwise, these lines of code could be removed by a print manager replacing the code between the %%BeginFeature and %%EndFeature comments.

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 keywords *OrderDependency and *NonUIOrderDependency, described in section 5.2, provide information about the appropriate setup section (described in the DSC) and relative ordering of each PostScript invocation. If a print manager is not coded to read the *OrderDependency and *NonUIOrderDependency statements in a PPD file, it must take care of the proper ordering of the code fragments by itself.

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 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 page region invocation (for example, Letter) occur, the media tray invocation should precede the imageable region invocation. Otherwise, the tray invocation might override the imageable region invocation.

The following items should occur after media size selection:

- job control requests such as duplex, automatic tray switching, signaturing, output bin selection, and finishing features such as folding, binding, and stapling

- halftone invocations (including halftone screen setup, transfer functions, and accurate screens). 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.

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 and their treatment is discussed in section 5.4.

Once a device is installed, features such as additional memory, paper trays, and fonts might be added to the device. In this specification, the term *system administrator* is used to mean the person who adds such features and is responsible for the maintenance of 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 some provisions for customizing the information contained in PPD files to adapt them to local instances of devices or to specific applications when necessary.

One approach to system management 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 utilities) can then query the user or the device or watch for system changes and update the database dynamically to reflect additional memory, fonts, available trays, and other changeable printer features.

A less dynamic approach is provided in this specification by *local customization files*, which contain only the changed or added items and a reference to the primary PPD file. In a given computing environment, there is usually one PPD file for each type or model of device in use. For example, there may be seven Acme FunPrinters in the system, but there is usually only one Acme FunPrinter PPD file, which is shared by or copied onto each host computer in the system. However, if applications or users want to add to or modify the contents of a PPD file, they can create a local customization file for a specific instance of a device or for use by a particular application.

For example, a computing environment might contain a primary PPD file that describes a generic Acme FunPrinter. If the FunPrinter in Room 13 has additional memory, and if the system does not provide utilities for querying the user or watching the state of the printer, then the system administrator might want to create a local customization file for the FunPrinter in Room 13 that reflects the presence of additional memory. (This is better accomplished by the use of *InstalledMemory in the primary PPD file, if the print manager supports it.) Or, if an application developer wants to add application-specific entries to a PPD file for a particular printer model, he would do so by creating a local customization file that would be used only by that application. For example, a color-intensive application might want to parse a PPD file for halftone information and add complementary halftone screens to a local customization file.

The local customization file should generally contain only entries only for items that are changed or added. However, to be understood by applications parsing PPD files, the local customization file must conform to the PPD specification, so in a sense, the local customization file is a minimal PPD file. The minimal set of required keywords listed in section 3.8 must be included at the beginning of the file, so print managers can recognize it as a PPD file. Other keywords that are marked *Required* in this specification, such as *PageSize, are not required in the local customization file, unless they are being customized.

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. Application developers can also create customization files with different extensions, which are read only by their application.

The local customization file must contain a reference to the primary PPD file in this format:

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

where *filename* is the name of the primary PPD file. This referencing allows a system administrator to later replace the primary PPD file without forcing users to edit their local customization files. If the new primary PPD file has the same name as the old one, it will automatically be referenced by the local customization file.

Before creating a local customization file, a system administrator should make sure that computing environment provides support for the concept. Some print managers might not process the *Include statement, or the system might not provide a way to install both the primary PPD file and the local customization file.

When a primary PPD file is included by a local customization 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 statement, possibly reducing the parsing time.

Because the first instance of a keyword is the correct instance, all keywords in a local customization file should occur *before* the *Include statement that references the primary PPD file.For example, assuming the primary PPD file is called *TIMICRO1.PPD*, a local customization file might look like this

```
*% Local Customization File for TI microLaser
*FreeVM: "1907408"
*Include: "TIMICRO1.PPD"
*% end of local customization file
```

The local customization file might be named *TICUSTOM.PPD*. A parser reading this file would record the value of *FreeVM as shown above, and would ignore subsequent occurrences of that keyword in the included PPD file, *TIMICRO1.PPD*.

Note *To application developers: The situation in the example above would be better handled through proper parsing of the *InstalledMemory keyword, eliminating the need for a local customization file. This example is intended for use with parsers that don't process *InstalledMemory but do process local customization files.*

If a UI keyword (see section 1.2) occurs in a local customization file, that keyword's entire entry must be present, to avoid confusing a print manager with partial entries. For example, to change the value of *DefaultManualFeed, the entire *ManualFeed entry must appear, including main keywords, options, query, and the *OpenUI/*CloseUI bracketing. It should also include any *OpenGroup/*CloseGroup bracketing, if the *ManualFeed entry is surrounded by *OpenGroup/*CloseGroup in the primary PPD file. (See section 5.2 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 (and subsequently ignore) multiple instances of a given *OpenUI/*CloseUI entry in the combination of the primary PPD file and any local customization files.

Note *The concept of "first instance is correct" does not apply to certain keywords that normally have multiple instances in a PPD file, and which do not have option keywords to distinguish those instances. For example, *UIConstraints and *PrinterError occur multiple times in a PPD file, with different values, but with no option keyword to distinguish one instance from another. In these cases, all instances must be parsed and recorded. This implies that a parser must either know the semantics of 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 A for a list of optionless keywords that might occur multiple times in a file.*

Using and Changing Default Settings

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

```
*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 the entire *ManualFeed entry, copied from the original PPD file, with the value of *DefaultManualFeed changed from False to True:

```
*OpenUI *ManualFeed: Boolean
*OrderDependency: 20 AnySetup *ManualFeed
*DefaultManualFeed: True
*ManualFeed True: "code"
*ManualFeed False: "code"
*?ManualFeed: "query code"
*CloseUI: *ManualFeed
```

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.

3 Format

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.

3.1 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 an error.
- Any byte code that is not in the following list is an error: decimal 32 through decimal 255 inclusive, plus decimal 9 (ASCII horizontal tab), decimal 10 (ASCII line feed), and decimal 13 (ASCII carriage return).

3.2 Main Keywords

All main keywords start with the leading special character * (decimal 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 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. However, there is no requirement for a *Default keyword to have corresponding main and query keywords in a PPD file. A *Default keyword may appear alone if it makes sense.

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 example, 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 statements will start at the beginning of a line. The * (asterisk) character that begins the main keyword in the statement 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 and slash, which serve as keyword delimiters. Note that space, tab, and newline are outside this range. 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>:  <optionn>
*<main keyword>  <option1>:  "PostScript language code"
*<main keyword>  <optionn>:  "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. By convention, all 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.
- Certain keywords are required to be present; see the beginning of section 5 for a list. For parsing, a chain of local customization files and included PPD files are considered one file, so most required keywords can appear

anywhere in the chain of files and do not have to be repeated in each file in the chain. (See section 2.6 for exceptions.) 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 statement (including multi-line code segments) should be skipped. However, read section 5.2 and 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 that 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 colon and slash.

- 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 3.3 for information on option keywords.

3.3 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 statement, 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. 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. See section 5.1 for information on keyword creation.

The option keywords currently known for each main keyword 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 created at any time. Documenting the option keywords is done to prevent redundancy in naming; it is not meant to restrict the list of option keywords available.

Option keywords may contain any printable ASCII characters within the range of decimal 33 to decimal 126 inclusive, except for the characters slash and colon, which serve as keyword delimiters. Note that space, tab, and newline are outside this range.

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

```
*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 3.5 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. For example:

```
*PageSize Letter
*PageSize Letter.Transverse
*PageSize Letter.2
```

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 qualifier. A serialization qualifier 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 qualifiers, 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 17x11. 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 (see section 5.2).

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, including any extensions or qualifiers separated by dots.

- 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. A slash indicates the presence of a translation string. If a translation string is present, it is terminated by a colon. White space and slashes are allowed in the translation string. A newline encountered before the colon should be considered an error.

3.4 Syntax of Values

The : (colon) character (decimal 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.

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 contains a syntactically correct PostScript language fragment that is usable by the PostScript interpreter. This allows an InvocationValue to be extracted from the PPD file and placed directly into the output file.

An InvocationValue meets the following conditions:

- Occurs only in statements where there is an option keyword present.
- Starts and ends with the double quote character " (decimal 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. Note that, unlike other values, a newline does not terminate an InvocationValue and a slash does not mark the beginning of a translation string.

- The following characters are forbidden between the starting and ending double quote characters:

  - byte codes outside the range of printable 7-bit ASCII (see section 1.2)

  - double quote character " (decimal 67)

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

QuotedValue

A QuotedValue meets the following conditions:

- Occurs only in statements **without** an option keyword, with one exception: `*JCL` keywords, which may have both an option keyword and a QuotedValue. See section 5.8 for information on `*JCL` keywords.

- Starts and ends with a double quote character " (decimal 67).

- Between the double quote characters, a QuotedValue consists of a sequence of literal and/or hexadecimal substrings (defined in section 1.2). A literal substring is a sequence of 8-bit byte codes, as defined in section 1.2, with the following characters forbidden:

  - double quote character " (decimal 67)

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

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

Note that a QuotedValue is the only type of value in a PPD file that can contain byte codes outside the range of printable 7-bit ASCII. Also, unlike an InvocationValue, forbidden literal substring characters in a QuotedValue can be represented as hexadecimal substrings, bounded by opening and closing angle brackets < (decimal 60) and > (decimal 62) as defined in section 1.2.

Note *To builders of PPD files: If you are concerned about the portability of a PPD file across different platforms (for example, Windows and Macintosh), you should also use hexadecimal substrings to represent any byte code that is out-*

side the range of printable 7-bit ASCII. Most PPD files are initially built in English, using only printable 7-bit ASCII, but when a PPD file is translated to another natural language, 8-bit byte codes may be needed. Such byte codes are often specific to the platform and to the natural language environment. If the file is being translated for a specific platform, the use of 8-bit byte codes will probably not be a problem for that platform's print managers, but if the PPD file is supposed to remain portable across platforms, the use of 8-bit byte codes may hinder portability.

A print manager parsing a QuotedValue is responsible for converting a hexadecimal substring into a sequence of bytes before using them.

- 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 statements both have valid QuotedValues:

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

*MainKeyword: "<ABCDEF>"
```

Note *To builders of PPD files: 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.*

Note *To application developers: 8-bit byte codes were not allowed in PPD files prior to the 4.3 specification, but this rule was widely violated when PPD files were translated to other natural languages. Most parsers simply pass the 8-bit byte codes along without rejecting them or attempting to translate them, but any parsers that expect only printable 7-bit ASCII may have problems with 8-bit byte codes in translated and newer PPD files.*

SymbolValue

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

A SymbolValue is a value that meets the following conditions:

- Starts with a caret ^ (decimal 94)

- Contains only printable 7-bit ASCII 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 .

StringValue

A StringValue can occur in a statement whether or not there is an option keyword present. 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 94), to avoid confusing a StringValue with a SymbolValue.
- The value is composed of printable 7-bit ASCII byte codes, possibly separated by spaces and tabs into multiple components. It is terminated by a newline, or by a slash, in the case of a translation string.
- There is no escape mechanism for forbidden characters.

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 a statement, and the first non-white-space character after the colon is a double quote, " (decimal 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 5.8 for a description of the `*JCL` keywords.
- If there is an option keyword in a statement, and the first non-white-space character after the colon is a caret, ^ (decimal 94), the value is a SymbolValue.

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

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

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

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

- The value of a query keyword (one that starts with the characters ***?**), although formatted as a QuotedValue, should be treated as an Invocation-Value.

- The value of a *Default keyword statement must be a StringValue.

- If a *Default keyword is not stand-alone (defined in section 1.2), the value must be a string matching a valid option keyword in the surrounding entry, or it may be Unknown. If the *Default keyword is stand-alone, the value must be one of the values registered in this specification for the main keyword that it would normally accompany, and the value may not be Unknown, as that provides no useful information.

- StringValues can contain spaces and tabs, 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 is 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 statement in the PPD file after the closing double quote delimiter. If it is not found, this is considered a *parse error* with a missing closing 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 statement, and hexadecimal substrings should not 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 before being used. Any non-hex data between the angle brackets is considered an error, as is an odd number of hex digits. White space and newlines between the angle brackets should be ignored.

- 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 statements and keywords.

3.5 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 a PPD file is translated into several languages, there will be one PPD file for each language. In various language versions of a PPD file, only the translation strings, certain QuotedValues that are used to identify the device, and possibly the comments, will differ. All other information, including main keywords and option keywords, will remain the same.

Note *To builders of PPD files: The values of the following keywords may be translated directly, without using a translation string: *Include, *ModelName, *NickName, *PCFileName, and *ShortNickName. Because these are QuotedValues, they may include 8-bit byte codes.*

A translation string can occur after any option keyword or after any type of value except a SymbolValue. The value of a *Default keyword may have a translation string only if it appears as a stand-alone keyword (see section 1.2), such as *DefaultColorSep, or *DefaultResolution without a corresponding *Resolution keyword. If the *Default keyword is not stand-alone (if it appears with an associated main keyword and option keywords), it does not need a translation string and should not have one, because such translation strings, if any, should occur with the option keywords, to avoid confusing parsers. See section 2.6, section 3.2, and section 4.5 for more information on *Default keywords.

With closely related statements, such as *PageSize and *PageRegion, it is impossible to predict which statement a print manager will read to get the translation string for an option keyword. For continuity of results, if a given option keyword has a translation string, and that option keyword is used with multiple main keywords and has the same semantics across those keywords, then the translation string should be on every occurrence of the option keyword and should be identical across occurrences. For example, if the *PageSize statement for Letter uses a translation string Portrait Letter, the *PageRegion, *PaperDimension, and *ImageableArea statements for Letter should all use the same translation string.

A translation string is detected by the presence of a slash (/) character (decimal 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 two cryptic values into strings that are more meaningful to the user (a “translation” into English):

```
*LanguageVersion: English
*DefaultColorSep: ProcessBlack.90lpi.1200dpi/90 lpi / 1200 dpi
*PrinterError: "CVR OPN"/cover open
```

Translation strings in a PPD file are optional. If translation strings are present, the translation strings should be used for display to the user, rather than displaying 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.

To unambiguously relate natural-language characters to byte codes, an encoding is specified for each natural language (such as English or French) that can be used in a PPD file. These encodings are described by the keywords *LanguageVersion and *LanguageEncoding, documented in section 5.3.

Translation strings may include 8-bit byte codes, such as characters with accents. See section 1.2 for the range of byte codes allowed in PPD files. If the builder of the PPD file is concerned about file portability across platforms, the byte code range in translation strings should be limited to printable 7-bit ASCII, with out-of-range byte codes represented by hexadecimal substrings (defined in section 1.2). A translation string can be represented partially or wholly as a hexadecimal substring. A print manager must convert the hexadecimal substring to the appropriate sequence of byte codes before displaying the translation string to the user.

The following example shows 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 1.2.
- The character colon : (decimal 58, if the translation string is on an option keyword.
- The characters < and > (decimal 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. 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 (defined in section 1.2), except that it cannot contain the following characters: newline, < (decimal 60) and > (decimal 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 1.2 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.

3.6 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.

3.7 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.

*End is used only when the code requires more than one line in the PPD file. In the following two examples, the *PageSize statement fits on one line and does not require *End. The *?Smoothing statement is an “extended” code sequence that does require *End:

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

*?Smoothing: "save
    [(None)(Light)(Medium)(Dark)]
    statusdict /doret {get exec}
    stopped { pop pop (Unknown)} if
    = 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.

Adobe recommends that the following syntax be used when building PPD files

```
<dict> /foo get exec
```

rather than

```
<dict> begin foo end
```

This will prevent errors caused by a print manager not cleaning up the dictionary stack properly after catching an error in a stopped context.

Level 2 PostScript language sequences

Sometimes a PostScript language file generated using a PPD file for a given device is redirected to another, different device. This can happen if the file is stored for later printing and the original device is not available at print time, or if files are exchanged between users with different printers, or if an intelligent spooler redirects the file to a more appropriate printing device. Although there is absolutely no guarantee that a PostScript language file created for a specific device will work on another device, there is a reasonable chance that it will, if the file contains few or no calls to special device features. One or more of the following may happen:

- The file causes the PostScript language interpreter in the device to abnormally terminate execution, possibly requiring the device to be reset.
- The file fails to print any pages.
- Features requested by the file, such as duplex or a certain size of paper, are not available, so these feature requests are ignored and the file prints on the device's default paper with default feature settings.
- Some requested features exist but their settings have different meanings on the current device, so the file prints but the results are unexpected. For example, the printing might be darker, or the image might be oriented sideways on the paper.
- The file prints correctly because all the feature requests are available and the settings have the same semantics, or because there are no feature requests and the default page size and default feature settings are acceptable.

A print manager can guard against the first two cases by executing each feature request in a stopped context. Thus, a request for a feature that does not exist on the current device will effectively be ignored. (See section 2.4 for more information on error handling by print managers.)

In the next two cases, the printed effect may not be exactly what the user requested, but at least they'll get something. For example, a file requesting duplex printing may be sent to a simplex printer, and depending on the print manager, it may print, but only on one side of each page.

To further the aim of printing whenever possible, even when Level 2 code is sent to a Level 1 device, the following recommendations should be followed when building a PPD file.

- Do not use the Level 2 dictionary syntax symbols **<<** and **>>** directly in invocation code when constructing dictionaries. Doing so will cause a **syntaxerror** if this code is re-directed to a Level 1 device. Such a **syntaxerror** cannot be trapped in a **stopped** context by a print manager. The two alternatives are to use the more verbose Level 1 method:

  ```
  N dict
  dup /name1 value1 put
  dup /name2 value2 put
  ...
  dup /nameN valueN put
  ```

or to put the more efficient Level 2 method into an executable string:

```
(<<) cvx exec /name1 value1 /name2 value2
...
/nameN valueN (>>) cvx exec
```

This second method will avoid the **syntaxerror** described above. It will consume a tiny amount of VM, which will be restored by automatic garbage collection on a Level 2 device.

3.8 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 5.2 for details on *PPD-Adobe.) Files conforming to this version of the specification would have the following statement:

```
*PPD-Adobe: "4.3"
```

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 *Required* in their individual descriptions in this document, and are listed at the beginning of section 5. By convention, the following *subset* of required keywords generally appears immediately after the copyright, in any order, except that *ShortNickname must occur before *NickName. 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	*Manufacturer	*ShortNickName

The number of bytes in a PPD file is **not** limited by this specification. However, certain print managers impose limits on some aspects of a PPD file. Builders of PPD files should be aware of the following restrictions:

- In the Windows environment, both PScript 4.x and AdobePS 4.x limit the combined total of all invocation code, query code, keywords, options, and translation strings in a PPD file to less than or equal to 64KB. Also, the number of *OpenUI and *JCLOpenUI entries must be less than 100.

- In the Macintosh environment, both LaserWriter 8.x and PSPrinter 8.x limit the size of each invocation code fragment to less than or equal to 32KB.

- In the Windows 95 environment, the Image Color Matching (ICM) system places additional restrictions on the content of *ModelName, *ShortNickName, and *Manufacturer in PPD files for color printers. See the descriptions of those keywords for details.

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*.

- Boldface type in prose denotes a PostScript language operator or dictionary key, such as the **setpagedevice** key **PageSize.**

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

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

  means that this statement in the PPD file will read either

  ```
  *DefaultManualFeed: True
  ```

  or

  ```
  *DefaultManualFeed: False
  ```

  but True and False cannot both appear in this statement.

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

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

  means that 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 statements are valid

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

UI Main keywords that are commonly bracketed with the *OpenUI/*CloseUI keywords in PPD files are marked in this specification with this symbol ("UI" for "user interface") next to the keyword definition. This does **not** mean that the main keyword **must** be bracketed by *OpenUI/*CloseUI; it only informs builders of PPD files that this keyword is commonly considered to be a UI keyword. See section 5.2 for details on *OpenUI/*CloseUI.

† A few of the main keywords may require starting an unencapsulated job (formerly known as "exiting the server loop") for correct execution. These have been flagged by a dagger in the left margin as shown here. If handling such a keyword, the print manager must perform the following tasks:

- declare an unencapsulated job using the DSC comment: %!PS-Adobe-3.0 ExitServer
- obtain a password from the user, from system software, or from the keyword *Password in the PPD file, and put the password on the operand stack
- emit the code from the daggered keyword
- end the job and use the DSC comment %%EOF

Builders of PPD files must ensure that the code contained in the value of a daggered keyword does the following:

- checks for the existence and validity of a password on the operand stack
- begins an unencapsulated job by using **exitserver** (Level 1) or **startjob** (Level 2)
- performs the function of the daggered keyword (for example, changes the resolution or fixes a bug)

The code to perform the first two functions can usually be copied from the value of the *ExitServer keyword in the PPD file.

4.2 Sample Keyword Statements

The format of section 5, which documents the individual keywords, 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 keyword looks like in this specification:

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

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

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

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

```
*Collate True: "(<<) cvx exec /Collate true
   (>>) cvx exec setpagedevice"
*End
*Collate False: "(<<) cvx exec /Collate false
   (>>) cvx exec setpagedevice"
*End
```

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

This indicates that for the keyword *MainKeyword, there can be many legal 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 statement 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: "(<<) cvx exec
   /PageSize [612 792] (>>) cvx exec setpagedevice"
*End
*PageSize Legal: "(<<) cvx exec
   /PageSize [612 1008] (>>) cvx exec setpagedevice"
*End
*PageSize Ledger: "(<<) cvx exec
   /PageSize [1224 792] (>>) cvx exec setpagedevice"
*End
*?PageSize: "query code to get current /PageSize"
*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"
```

***?MainKeyword:** "*query*" (returns: *keywordOption* | Unknown)

This is a query keyword, evidenced by the *? prefix. Its value is a *query*, as defined in section 4.3. The valid return values for this query are documented in this specification to the right of the sample query statement; they are not part of the statement itself. That is, the PPD file would contain:

```
*?MainKeyword: "code to perform query"
```

***MainKeyword** *option*: "*invocation*"

***DefaultMainKeyword:** *keywordOption* | Unknown

***?MainKeyword:** "*query*" (returns: *keywordOption* | Unknown)

Where it makes sense, the main keyword, default keyword, and query keyword are documented in a block together, as shown here. The valid return values for the query keyword are documented to the right of the sample query statement.

4.3 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, represented as a hex string. 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 special case of a *string*, which is defined in this section. 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 font name 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 4.295×10^9 (32 bits).

invocation

An *invocation* is a 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 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 8-bit byte codes.

option

An *option keyword*, or simply *option*, is a *string* subject to the rules defined in section 3.1 and section 3.3. In this specification, the placeholders for option keywords are generally preceded by a descriptive qualifier, as in *mediaOption* or *trayOption*.

query

A *query* is a sequence of valid PostScript language commands that requests information from the device and returns a string, terminated by a newline, to the host via the reverse channel of the device. Translation strings are not allowed on the values returned from queries. Valid return values for a query are determined as follows:

- If the associated main keyword is a UI keyword (defined in section 1.2), the query must return one of the option keywords listed in the PPD file for that main keyword, or it can return the string Unknown. For example, if *PageSize is a UI keyword, then the *?PageSize query can only return valid page size options (such as Letter and Legal, without translation strings) that are listed under *PageSize, or it can return Unknown, terminated by a newline.

- If the associated main keyword is not a UI keyword, then the valid return values for the query keyword are documented in this specification. For examples, see *?FontQuery, *?FontList, *?ImageableArea, *?InsertSheet, and *?FileSystem. The return value must be terminated by a newline.

Note *To builders of PPD files: Because of its format, the value of a query is a QuotedValue, but you should follow the rules for InvocationValues when writing queries, as the query must be a valid PostScript code fragment.*

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 3.4×10^{-38} to $3.4 \times 10^{+38}$. A *real* in a PPD file will not include scientific or exponential notation. That is, the following are all valid:

```
1.4 -4.2273 .165 0.165
```

but the following are not valid:

```
1.4e-4 -1.45E7 -1.45E+7
```

string

A *string* is comprised of any printable characters, but is delimited by white space. The length of a *string* is limited by the 255 characters-per-line limit described in section 3.1.

text

A *text* string may contain any characters that are legal in the value type. For example, *text* in a QuotedValue may contain any character that is legal in a QuotedValue, including spaces, hexadecimal substrings, and 8-bit byte codes. Individual keywords may further restrict the allowable characters in a *text* string, so *text* strings cannot be treated uniformly by a print manager. The length of a *text* string is limited by the 255 characters-per-line limit described in section 3.1.

4.4 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 statement 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 statement 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 statement 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 deselect the feature. For example, the following statement contains the code to deselect the manual feed slot.

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

None

Like False, None is used to indicate that a certain feature is deselected (off) by default, and also to indicate how to deselect (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 turn off the feature. In the example above, the None option would contain code to 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 and *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 a value for *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). However, a stand-alone *Default keyword should never have a value of Unknown, as that provides no useful information for a print manager.

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.*

4.5 Summary of Rules for *Default Keywords

Because information about *Default keywords is distributed throughout the preceding part of this document, this section summarizes the rules for *Default keywords, both for print manager authors and for builders of PPD files.

- A *Default keyword may appear as part of an entry or by itself. A *Default keyword appearing by itself is referred to as a *stand-alone* *Default keyword (see section 1.2). Some *Default keywords are always stand-alone.

- The value of a *Default keyword must be a StringValue (no quotes).

- If a *Default keyword is part of an entry, its value must be one of the options of the main keyword in the entry, or it may be Unknown.

- If a *Default keyword is stand-alone, its value must be one of the options registered in this specification for the main keyword that it would normally accompany, or the *Default keyword must have its own set of valid values documented in this specification and the value must be one of those. A stand-alone *Default keyword may not appear with the value of Unknown, as that provides no useful information to a print manager.

- A stand-alone *Default keyword may have a translation string on its value.

- A *Default keyword that is part of an entry may not have a translation string on its value, because it is likely that translation strings will occur on the option keywords of the main keyword in the entry, and a print manager would not know which translation string to use.

5 Keywords

This section contains a description of how builders of PPD files can define their own main keywords, followed by documentation 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 prevent a given combination of main keyword and option keyword from having multiple meanings across multiple PPD files.

All keywords are optional in a PPD file, unless noted as *Required* in the keyword description. The following keywords are currently required:

*DefaultImageableArea	*DefaultPageRegion	*DefaultPageSize*
*DefaultPaperDimension	*FileVersion	*FormatVersion
*ImageableArea	*LanguageEncoding	*LanguageVersion
*Manufacturer	*ModelName	*NickName
*PageRegion	*PageSize	*PaperDimension
*PCFileName	*PPD-Adobe	*Product
*PSVersion	*ShortNickName	

Note **Manufacturer is new as of the 4.3 version of this specification. *ShortNickName is newly required in the 4.3 version; it was not required in previous versions.*

Every feature of the device that can be described by a PPD keyword should be included in the PPD file. Inclusion of all relevant keywords in the PPD file produces the most complete picture of the device for the print manager and user. Builders of PPD files should consider all non-required keywords to be recommended for inclusion in the PPD file if they are relevant to the device.

On the other hand, if a feature is not supported by a device, that feature's default, invocation, and query keywords **must** be omitted from the PPD file. To a print manager, absence of a feature in the PPD implies lack of device support for that feature.

5.1 Creating Your Own Keywords

Device manufacturers and other builders of PPD files may create their own main keywords by following the rules in this specification. For tracking purposes, such keywords must be named with a prefix unique to the creator. For example, Acme Printer Co. might use the prefix "*AC", and their keywords would have names like *ACHalftone. The same prefix must be used on all keywords created by one company. For ease of tracking and guaranteed prefix uniqueness, Adobe recommends using the same two-letter prefix already in use by the creator for their PPD file names (and *PCFileName). The list of assigned prefixes can be found in *Appendix D: Manufacturer's Prefix List and *Manufacturer Strings*, and updates can be obtained from the address on the front cover.

Keywords created outside of Adobe must not conflict with or duplicate the definition of keywords already existing in this specification. To avoid unnecessary proliferation of keywords, builders of PPD files should make every effort to see if their device's feature can be defined by an existing keyword, before creating a new keyword.

Print managers cannot be expected to know how to deal with an unknown keyword, unless the keyword is enclosed by the *OpenUI/*CloseUI construction and follows all the rules associated with that construction. See section 5.2 for information about *OpenUI and *CloseUI.

Builders of PPD files may also create their own option keywords if necessary. No unique prefix is necessary for option keywords, but all rules pertaining to option keywords must be followed. Again, every effort should be made to use an existing option keyword before defining a new option keyword. See section 3.3 for rules pertaining to option keywords. In addition, although it is not required, option keywords usually have the first letter capitalized, for better readability of the user interface created by a print manager.

5.2 Structure Keywords

The keywords in this section do not invoke any device features. Instead, they provide structure to a PPD file and are intended to aid in parsing the PPD file and building a user interface.

***PPD-Adobe:** "4.3"

Required. This keyword must be the first line in a valid PPD file. The QuotedValue of "4.3" signifies that this file conforms to the 4.3 version of this specification. Conformance to a later version of this specification would be indicated by a higher number. A parser can assume that a changed digit to the right of the decimal indicates a minor revision to the specification, and the file can be safely parsed by older parsers. A change in the digit to the left of the decimal indicates a more significant change in the specification, and a file that conforms to the newer version of the specification may be incompatible with older parsers.

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

***CloseUI:** *mainKeyword*

These keywords allow a parser to differentiate between UI keywords, which would typically be displayed in the user interface, and informational keywords, which would typically not be displayed. *OpenUI and *CloseUI are used to bracket all the information about a UI keyword, if that 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. (Adobe strongly recommends that every *OpenUI/*CloseUI entry contain an appropriate *OrderDependency statement, so that the print manager knows where to emit the code in the output file.)

The StringValue values PickOne, PickMany, and Boolean denote the type of option list 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 deselected. 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.

  The option None has special meaning. Code associated with None explicitly deselects the other options in the list. For example, a list of stapling options might include None, meaning "do not staple." If the user selects None, the print manager should send 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 code associated with None explicitly deselects all other options in the list. This is an exception to the general PickMany rule of allowing more than one choice at a time. In a user interface, if a user selects None, all other options for that feature 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. This tells a print manager that this feature could be displayed as a check box or radio buttons, rather than as a menu list.

The parameter *mainKeyword*, which is both the option of the *OpenUI statement and the StringValue of *CloseUI, is the keyword whose options are listed within the *OpenUI/*CloseUI pair. The value may optionally have a translation string to show the manufacturer's preferred name for the feature.

Here 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, checkbox, or radio buttons labeled "Smooth Characters" with two options, True and False.

The *OpenUI/*CloseUI structure provides several benefits:

- It allows a print manager to build a user interface automatically and uniformly. The user-selectable options bracketed by *OpenUI/*CloseUI follow a certain form. Each entry contains one or more statements consisting of a main keyword, option keyword, and a value, which is a code sequence. The print manager can display the options, record the user's selection of an option, extract the code for that option, and send the code to the device, all without having any semantic knowledge of the feature and its options.

- New options can be added to a main keyword without the print manager having to be rewritten to parse for them. Assuming that a print manager simply displays the options and executes the associated code, the print manager does not need to recognize the option or know anything about the code it is sending to the output file.

- 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 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.

UI

Main keywords that are commonly bracketed with *OpenUI/*CloseUI are marked in this specification with the UI symbol (shown here) next to the keyword definition. Some features that appear in the user interface require extra work from the print manager, 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 entries will not be bracketed with *OpenUI/*CloseUI, even though they might appear in the user interface via special handling by the print manager.

The keywords *JCLOpenUI and *JCLCloseUI provide the same structure for JCL keywords. See section 5.8 for their description.

*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. A selection of features can be grouped by placing a *OpenGroup/*CloseGroup pair around the feature set. If nested groups are needed (groups within groups), *OpenSubGroup and *CloseSubGroup must be used. *OpenGroup/*CloseGroup may not appear inside the bracketing of another *OpenGroup/*CloseGroup pair.

Features in 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 features that do not exclude each other.

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 group name string must conform to the rules for the elementary type *string* (see section 4.3) and it must be unique; there can be only one group or sub-group with a given name.

The example below groups all of the finishing features of the device (the invocation code is omitted to save space, and the blank lines are only for readability):

```
*OpenGroup: Finishing

*OpenUI *FoldType: PickOne
*DefaultFoldType: None
*FoldType Saddle: " "
*FoldType None: " "
*CloseUI: *FoldType

*OpenUI *Collate: Boolean
*DefaultCollate: True
*Collate True: " "
*Collate False: " "
*CloseUI: *Collate

*CloseGroup: Finishing
```

The sample PPD file in section 6.2 shows imagesetter features in a group. Another logical group is finishing features. By parsing for the keywords *OpenGroup/*CloseGroup, a print manager can display Finishing and Imagesetting to the user as buttons or menu items, and all the finishing features and imagesetting features can be 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.

***OpenSubGroup:** *string*

***CloseSubGroup:** *string*

These keywords are used to designated groups within groups. They can only appear within a *OpenGroup/*CloseGroup bracketing keyword pair. Groups may be nested to any level with *OpenSubGroup/*CloseSubGroup. The group name string must conform to the rules for the elementary type *string* (see section 4.3) and it must be unique; there can be only one group or sub-group with a given name. See the description of *OpenGroup/*CloseGroup in this section.

Here is an example of the syntax of these keywords (be aware that the *OpenUI/*CloseUI entries are not syntactically complete in this example):

```
*OpenGroup: ProductionPrinting

*OpenSubGroup: MediaSelection
*OpenUI *PageSize ...
...
*CloseUI: *PageSize
*OpenUI *MediaType ...
...
*CloseUI: *MediaType
*CloseSubGroup: MediaSelection

*OpenSubGroup: Administration
*OpenUI ...
...
*CloseUI
*OpenSubGroup: TopSecret
...may include several *OpenUI/*CloseUI entries
*CloseSubGroup: TopSecret
*OpenUI ...
...
*CloseUI
*CloseSubGroup: Administration

*CloseGroup: ProductionPrinting
```

The sub-group TopSecret is nested within the sub-group Administration, which is nested within the group ProductionPrinting. Note that the Administration group contains a mix of *OpenSubGroup and *OpenUI entries at the same level.

The result of this might be that the print manager would display a panel with a button or menu item labeled ProductionPrinting. Selecting ProductionPrinting would produce a menu or a button panel advertising MediaSelection and Administration. Selecting MediaSelection would display two *OpenUI choices *PageSize and *MediaType for the user. Selecting Administration would display two *OpenUI choices and a button or menu item labeled TopSecret, which, when selected, would display more buttons or menu items.

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

***NonUIOrderDependency:** *real section mainKeyword optionKeyword*

These keywords provide 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 information about why order dependencies exist.

*OrderDependency is used only with UI keywords (keywords surrounded by the *OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI keywords). *OrderDependency must occur inside the *OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI bracketing for that keyword.

*NonUIOrderDependency is used only with the few non-UI keywords that emit code. For example, *CustomPageSize is not a UI keyword because it requires user input, which is not accommodated by the *OpenUI choices of PickOne, PickMany, and Boolean. Yet *CustomPageSize typically would appear in a user interface specially built by a print manager, and when selected by the user, *CustomPageSize emits code fragments. A print manager needs to know the order in which these code fragments should be emitted, relative to all other code fragments. *NonUIOrderDependency provides that information for non-UI keywords.

Although *OrderDependency and *NonUIOrderDependency are not required keywords, to ensure correct results during printing, Adobe strongly recommends that every *OpenUI/*CloseUI entry and every *JCLOpenUI/*JCLCloseUI entry contain a *OrderDependency statement and that all non-UI keywords that emit code have an associated *NonUIOrderDependency statement.

The value is a StringValue with multiple components separated by white space. Each component is defined identically for both keywords. The value of *real* specifies the relative order in which this code should be emitted. It defines the ordering across all device-specific code fragments within a *section* (defined later). The absolute values of the order numbers are not important, only their values relative to other order numbers within the same *section*. Within a *section*, code assigned a lower number should be emitted before code assigned a higher number. Multiple code fragments in a single *section* can be assigned the same order number. If the order numbers assigned to two code fragments are equal, the code fragments can be emitted in either order. Code fragments that do not have an *OrderDependency or *NonUIOrderDependency statement should be emitted after all fragments that do have ordering numbers.

The value of *section* 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 that will take effect at the start of the next job. This code should be sent as a separate job, before the job 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 and must follow the rules defined for the values DocumentSetup or PageSetup accordingly.

Note *To builders of PPD files: Use* AnySetup *as the* section *name if there is no specific reason to specify* PageSetup *or* DocumentSetup*. This lets a print manager make the most efficient decision about where to emit the code fragment. However, to be labeled* AnySetup*, the code fragment must work correctly when emitted in either section,*

The values of *mainKeyword* and *optionKeyword* specify a statement in the PPD file that emits device-specific code, which must be emitted in the order defined by *section* and *order*. The parameter *optionKeyword* will be omitted if *mainKeyword* 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 (this is the common case).

The following example specifies that the code to change the resolution must be emitted before the code to change the input slot (these entries are not syntactically complete):

```
*OpenUI *Resolution: PickOne
*OrderDependency: 10 AnySetup *Resolution
*DefaultResolution: 1200dpi
*Resolution 2504dpi: "..."
*Resolution 1200dpi: "..."
*?Resolution:"..."
*CloseUI: *Resolution

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

Note *Builders of PPD files for Level 2 devices can facilitate optimized performance by assigning the same *OrderDependency section and order number to as many keywords as possible. A print manager can then bundle all of the code fragments that have the same values for section and order into one **setpagedevice** call, by redefining **setpagedevice** to collect the key-value pairs into a single dictionary that is then executed with a single real **setpagedevice** call. Very few keywords require a different order number on Level 2 devices, so keywords should be assigned the same order number unless there is some reason to assign a different order number. If a given code fragment would cause different results when executed alone than it would when bundled with other parameters (in any order) into a single **setpagedevice** call, that code fragment should be assigned a separate order number. For example, if the code fragments for *ManualFeed and *Duplex are both emitted in the AnySetup section, and the result of emitting the code fragments separately, like this:*

```
<</ManualFeed true>> setpagedevice
<</Duplex true>> setpagedevice
```

produces the same result as emitting the code fragments together, like this:

```
<</ManualFeed true /Duplex true>> setpagedevice
```

or in a different order, like this:

```
<</Duplex true /ManualFeed true>> setpagedevice
```

*then *ManualFeed and *Duplex can be given the same order number. Otherwise, if any one of these methods gives a different result than the other methods, *ManualFeed and *Duplex should be assigned different (and appropriate) order numbers.*

Code fragments ordered by either *NonUIOrderDependency or *OrderDependency should be considered to be a single set; that is, their order numbers within a section are relative to each other. For example, a code fragment with a *NonUIOrderDependency order number of 20 and a *section* of PageSetup would be emitted before a code fragment with a *OrderDependency order number of 30 and a *section* of PageSetup.

***QueryOrderDependency:** *real mainKeyword optionKeyword*

If there are device queries accompanying a print job, all such queries must be emitted in a “query job”, which immediately precedes the print job. Among other things, this allows spoolers to process the queries without processing the actual print job. (See the *DSC* for a description of query jobs.) In some cases, the query job must also contain the invocation code from certain main keywords, emitted before certain queries, in order for the query response to be accurate. For example, changing the resolution of the device can affect the amount of free VM, so to obtain a more accurate estimate of free VM, the device resolution code should be emitted in the query job before querying for free VM.

*QueryOrderDependency identifies main keywords whose invocation code needs to be emitted in the query job, before any queries, in order for the query response to be accurate. If a print manager queries for any of the following items: free VM, device resolution, available fonts (and there may be others), any invocation code referenced by *QueryOrderDependency should be emitted in the query job before any of these queries are emitted.

The structure of *QueryOrderDependency is very similar to the structure of *OrderDependency, although there is no *section* parameter because the entire query job is considered to be a single section. The value is a StringValue with multiple components separated by white space. The parameter *real* specifies the relative order in which this invocation code should be emitted within the query job. Code with a lower order number should be emitted before code with a higher order number. If two main keywords have the same *QueryOrderDependency order number, their code fragments may be emitted in either order relative to each other. All invocation code from main keywords flagged by *QueryOrderDependency should be emitted before any queries are emitted.

The parameter *mainKeyword* will be a valid UI keyword (defined in section 1.2) or a valid non-UI keyword can be constrained by *NonUIConstraints (see the description of *NonUIConstraints for a list of legal keywords). The parameter *optionKeyword* will be a valid option for that main keyword. If the main keyword has no options, the parameter *optionKeyword* will be omitted. The parameter *optionKeyword* may be omitted if all of the options for the main keyword would otherwise have the same order number (this is usually the case). The *QueryOrderDependency statement for a given UI keyword should be within the *OpenUI/*CloseUI bracketing for that keyword. For example:

```
*OpenUI *Resolution: PickOne
*OrderDependency: 10 AnySetup *Resolution
*QueryOrderDependency: 10 *Resolution
*Resolution 600dpi: "..."
*Resolution 1200dpi: "..."
*?Resolution: "..."
*CloseUI: *Resolution
```

This tells the print manager that the invocation code for the resolution chosen by the user should be emitted in the query job, before any queries, and then emitted again in the print job. There is no relationship between the order numbers of *OrderDependency and *QueryOrderDependency.

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

This keyword, whose value is a StringValue with multiple components separated by white space, denotes exclusionary relationships between pairs of 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 between the A4 page size and all input slots other than the upper slot.

*UIConstraints can also be used to express the relationship between an optional hardware component and the user-selectable features it enables. For example, the presence of an envelope feeder would enable printing on envelopes, 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. See section 5.4 for a description of the InstallableOptions group and how it interacts with *UIConstraints.

A print manager can use the information found in a *UIConstraints statement to make constrained choices unavailable to the user, either by “graying out” or removing these items from the print panel, or by some other method appropriate to the user interface. To obtain consistent behavior from print managers, *UIConstraints may only be used between pairs of UI main keywords (keywords whose entries are surrounded by the *OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI keyword pairs). Since all UI keywords are presented to the user as choices in a user interface, their behavior is consistent; when they are constrained, that choice should not be offered to the user. For information on constraining non-UI keywords, see the description of *NonUIConstraints in this section. If either of the keywords in the constraint pair is a non-UI keyword, you must use *NonUIConstraints.

The syntax of *UIConstraints 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 UI features are reciprocal, so there will be two statements for each pair of features. One statement tells you that FeatureA-Option1 constrains FeatureB-Option5, and the other statement tells you that FeatureB-Option5 constrains FeatureA-Option1. The PPD file builder must ensure that both statements appear, for the benefit of print managers that do not enforce automatic reciprocity. If one of the reciprocal statements is missing, a print manager should operate as if it were present.

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.

Each feature or both features may also 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: *StapleWhen *MediaType Transparent
```

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

Likewise, this format is legal:

```
*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.

Print managers must be careful to avoid permanently excluding the selection of a feature that is constrained by other features. For example, if selecting a page size of A4 disables the selection of the Lower input slot, how will the user select the Lower input slot when they want to switch to a different page size? The user could try selecting each page size, hoping to find one that works with the Lower slot. A better choice would be for the print manager to allow the selection of constrained items, but to automatically modify the other component(s) of the constraint (and inform the user), or to warn the user of the conflict and ask the user to resolve it.

Note *To builders of PPD file: *UIConstraints is used only between pairs of main keywords. Do not write constraints between the various options of a single main keyword. For example, do not write a constraint between *PageSize Letter and *PageSize Legal. Options are already constrained against each other by their PickOne or Boolean type, so additional constraints are unnecessary and illegal.*

***NonUIConstraints:** *keyword1 option1 keyword2 option2*

This keyword works exactly like *UIConstraints except that it is used with non-UI keywords (keywords not surrounded by *OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI). Please see the description of *UIConstraints for the complete syntax and semantics of *NonUIConstraints. All of the rules specified for *UIConstraints apply to *NonUIConstraints, except where noted in this description.

*NonUIConstraints must be used if *either* the constraining or constrained keyword is a non-UI keyword. For example, here is a UI keyword constraining a non-UI keyword:

```
*NonUIConstraints: *InputSlot Upper *CustomPageSize True
```

This indicates that if the user has chosen the Upper input tray, a custom page size cannot be requested.

The semantics of *NonUIConstraints are that a constrained feature is "not available". Because non-UI keywords are not consistent in behavior, the exact action taken is up to the print manager. Advice to the print manager encountering a constrained non-UI keyword might be "don't do what you would normally do". For example, for keywords such as *CustomPageSize and *InsertSheet, which require a print manager to display a specialized user interface, that interface should not be displayed or should be grayed out.

The following keywords are the only non-UI keywords that can appear in a *NonUIConstraints statement: *CustomPageSize, *LeadingEdge, *UseHWMargins, *InsertSheet, *FaxSupport, and *SetResolution. More keywords may be added to this list in future versions of this specification, after an analysis to determine the impact on existing applications.

Note *To application developers: In a future edition of this specification, *Font will be added to the list of keywords which can be constrained by *NonUIConstraints. See the description of *Font for a discussion of how application authors should handle this pending change.*

*NonUIConstraints are **not** automatically reciprocal. If the reciprocal constraint makes sense, it will be included in the PPD file, and a print manager should honor it, but if a reciprocal constraint is missing, a print manager should not attempt to create and enforce such a constraint.

***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 4.3 for details on the syntax of *filename* and section 2.6 for details on including one PPD file within another.

***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.

5.3 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, including upgrading the PPD file to match a new version of the *PPD File Format Specification*, 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 and from 1.9 to 1.10). 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 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." For a PPD file to conform to the version of the specification detailed in this document, the value of *FormatVersion must be "4.3". If a PPD file is updated to reflect changes in the *PPD File Format Specification*, this statement should be changed to match the new specification version number.

***LanguageEncoding:** *encodingOption*

Required. This keyword complements and partially supersedes the older *LanguageVersion keyword. *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
- ISOLatin2—Uses the ISOLatin2 encoding
- ISOLatin5—Uses the ISOLatin5 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. See section 3.5 for details on translation string syntax.

Note *To builders of PPD files: If the initial PPD file is built in English, the value of *LanguageEncoding is ISOLatin1. If you are building the file in another language, or if you are translating the file to another language, change the values of *LanguageVersion and *LanguageEncoding to reflect that language.*

***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 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
Turkish	varies; check value of *LanguageEncoding

***Manufacturer:** *"text"*

Required. This QuotedValue provides the name of the company that manufactured the device. If a device is OEM'ed from one manufacturer to another, this name will be the name of the manufacturer who is marketing the device under their own name. A given manufacturer must use the same name string in each of their PPD files. For example, the name must not be "Acme Printer Co." in one PPD file and just "Acme" in another. Also, the string must be unique among manufacturers; two or more manufacturers may not use the same string. Appendix D provides the current list of known manufacturer strings.

This keyword gives print managers a way to group together several devices provided by one manufacturer. Given a set of PPD files, a print manager can provide a list of manufacturers for the user to choose from, and then provide a list of PPD files available from the chosen manufacturer.

Note *To builders of PPD files: For ease of sorting and display by a print manager, we recommend leaving out unnecessary words like "Incorporated," "Company," "Ltd.", and "Corporation" when creating this string. The purpose of this keyword is for a print manager and a user to be able to distinguish one manufacturer from another, not to provide a complete account of the manufacturer's full legal name.*

Note *To builders of PPD files: In order for the Windows 95 Image Color Matching (ICM) system to match InterColor Consortium (ICC) color characterization profiles to devices and drivers, the first 4 characters (or until a space or hyphen is found, whichever comes first) of *Manufacturer must be a unique string, which must match the value of the icHeader.manufacturer tag in the ICC profile. Practically speaking, this means that in addition to the entire string being unique among manufacturers, the first 4 characters of *Manufacturer must also be unique, at least for color printers that have ICC profiles in the Windows 95 system. See the Microsoft document "Image Color Matching and Printer Drivers in Windows 95" for details on how color profiles are matched to devices and drivers. See Appendix D for a list of currently known icHeader.manufacturer profile tags.*

***ModelName:** *"text"*

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 must be unique for a given model of device. Because *Product is not always unique or descriptive, *ModelName is used by print managers to identify a PPD file for a specific device model. For example:

```
*Product: "(LaserPrinter)"
*ModelName: "Acme SuperPrinter Turbo v2011.108"
```

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 characters, whose decimal values are defined here or in section 1.1: alphanumeric characters, space, period, slash, hyphen (decimal 45), and plus (decimal 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 statement 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"
```

If the product is a PostScript language cartridge or other external add-on device, include that fact in the name. For example, *"Acme PostScript Cartridge for FunPrinter III"*.

Note *To builders of PPD files: As a guideline,* *ModelName *should contain the manufacturer's name followed by the device's name. In order for the Windows 95 Image Color Matching (ICM) system to match InterColor Consortium (ICC) color characterization profiles to devices, the* *ModelName *string must exactly match the string used to generate the value of the* icHeader.model *tag in the ICC profile. In addition, under certain circumstances some print managers assume that the first 4 characters (or until a space or hyphen is found) of* *ModelName *will be the name of the manufacturer and will match the* icHeader.manufacturer *tag in the ICC profile. Practically speaking, this means that the first 4 characters of* *ModelName *must match the first 4 characters of* *Manufacturer, *at least for color printers that have ICC profiles in the Windows 95 system. Finally, because some print managers use* *ShortNickName *instead of* *ModelName *for ICC profile matching, and the builder of the PPD file doesn't know which print manager will be used,* *ModelName *should match* *ShortNickName *to ensure consistent ICC profile matching. See the Microsoft document "Image Color Matching and Printer Drivers in Windows 95" for details on how color profiles are matched to devices and drivers and how the* ic.model *profile tag is generated.*

***NickName:** *"text"*

Required. This 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).

There may be only one *NickName in a PPD file. If the PPD file is valid for more than one product, that fact must be reflected in the *NickName value, as in *ModelName. Initially, the value of *NickName is usually the same as the value of *ModelName, but it can be edited in a local customization file if necessary, whereas *ModelName should not be changed. Alternatively, the value may have a translation string for localization.

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.

The length of the value of *NickName is unrestricted, except for line length. For situations where the length of the nickname must be restricted, see the description of *ShortNickName.

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

Required. This value, a QuotedValue, provides the name of the PPD file as it would appear in a PC environment. This name must be 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.

The file naming scheme employed by Adobe Systems for these PPD files is an attempt at a mnemonic name as constricted by the DOS operating system. The naming scheme is partially algorithmic. Filenames and the keyword *PCFileName are constructed as follows:

- The first 2 characters designate the manufacturer. A list of manufacturer prefixes is kept by Adobe Systems to ensure uniqueness of names. Please refer to *Appendix D: Manufacturer's Prefix List and *Manufacturer Strings* for the current prefix list.
- The next 6 characters consist of a string that uniquely identifies the device model.
- The suffix of the file is ".PPD".
- For consistent readability and to minimize confusion of letters with numbers, only capital (uppercase) letters are used.

Note *To builders of PPD files: If a released PPD file is changed, but the product itself has* ***not*** *changed, the PPD file name (and *PCFileName) will* ***not*** *change. Examples of this type of change to a PPD file include, but are not limited to, fixing a typographical error in the PPD file, fixing an incorrect value for *FreeVM, fixing feature code that was not tested properly and has been found to be incorrect, adding or changing translation strings, changing the names of option keywords, removing a keyword that was never supported by the device, and fixing incorrect information in a *Font statement.*

If a released PPD file is changed because the product itself has changed, the upgraded product must be issued a new PPD file with a new name. Any new features must be thoroughly tested. If the "old" and "upgraded" products are substantially the same product, marketed under similar names, Adobe recommends keeping the filenames substantially the same, but using the 8th character of the new filename as a version number for the file. For example, if the Acme SpiffyPrinter is upgraded with more memory, you might change ACSPIFFY.PPD to ACSPIFF2.PPD. Using the 8th character as a version number is a recommendation of common practice, but is not a requirement of this specification. If the "old" and "upgraded" products are substantially different, or marketed under different names, you should give the new PPD file a unique name that corresponds as closely as possible to the name under which the product is marketed.

Examples of filenames constructed according to the naming scheme:

APLWIIG1.PPD	Apple LaserWriter IIg v2011.113
DPLZ9601.PPD	Dataproducts LZR 960 v2010.105
OKOL8701.PPD	Okidata OL870 v2013.108
TKPHZR21.PPD	Tektronix Phaser II PXi v2010.116
TKPHZR22.PPD	Tektronix Phaser II PXi v2011.108
XR_88121.PPD	Xerox 8812 v2012.016

When a PPD file is translated into another language, Adobe recommends that you keep the same file name, but isolate the different language-version files from each other through directory structures or other physical isolation techniques. Different language versions of PPD files are essentially interchangeable at the software level, but need to be organized separately at the user level.

***Product:** *"(text)"*

Required. This 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. For example, if two devices have identical PPD files, they can be combined into a single PPD file with the following two statements

```
*Product: "(LaserPrinter)"
*Product: "(LaserPrinterII)"
```

and *ModelName, *NickName, and *ShortNickName would reflect the fact that the PPD file is valid for two products, like this:

```
*ModelName: "Acme LaserPrinter or LaserPrinterII"
```

***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 if the PPD file is valid for more than one version (and revision) of the interpreter. For example, if, for a given device, the PPD files for interpreter version 2011 revision 108 and interpreter version 2011 revision 120 are identical, they can be combined into one PPD file with the following statements:

```
*PSVersion: "(2011) 108"
*PSVersion: "(2011) 120"
```

Matching a PPD file to a device request

*PSVersion can be used by a print manager to match a particular version of a PPD file to a request for that file. However, in general, a PPD file for an older version of an interpreter can be safely used with a newer version of the interpreter, and vice versa. Interpreter upgrades are often based on performance and bug fixes that don't affect the PPD file. 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, separate PPD files should be built to describe each controller-engine combination. However, the product name, version, and revision number all describe only the controller, and thus would 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 S7000. The SuperRIP controller contains interpreter version 2012, 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 S7000 engine. These PPD files might be called, respectively, *SU2500_1.PPD* and *SU7000_1.PPD*.

The relevant statements in *SU2500_1.PPD* would look like this:

```
*PSVersion: (2012) 1
*Product: "(SuperRIP)"
*NickName: "SuperRIP with S2500 v.2012"
```

and the relevant statements in *SU7000_1.PPD* would look like this

```
*PSVersion: (2012) 1
*Product: "(SuperRIP)"
*NickName: "SuperRIP with S7000 v.2012"
```

These two PPD files, while matching in *Product and *PSVersion, are differentiated by their *NickName statements and their filenames. A print manager trying to choose the correct PPD file for a device must take all these things into account, or it can simply ask the user to select the correct file.

***ShortNickName:** *"text"*

Required. This keyword is identical to the semantics and syntax of the *NickName keyword, but the length of the string value is limited to 31 or fewer characters. Common practice is to make it the same as *NickName, but without the PostScript interpreter version number. 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. There may be only one instance of *ShortNickName per PPD file. Due to limitations in certain print managers, *ShortNickName must appear in the PPD file before *NickName.

Note **ShortNickName was not required in previous versions of this specification. It is required in files that conform to the 4.3 specification.*

Note *To builders of PPD files: As a guideline, *ShortNickName should contain the manufacturer's name followed by the device's name. In order for the Windows 95 Image Color Matching (ICM) system to match InterColor Consortium (ICC) color characterization profiles to devices, the PScript 4.x print manager requires that the *ShortNickName string exactly match the string used to generate the value of the icHeader.model tag in the ICC profile. In addition, under certain circumstances some print managers assume that the first 4 characters (or until a space or hyphen is found) of *ShortNickName will be the name of the manufacturer and will match the icHeader.manufacturer tag in the ICC profile. Practically speaking, this means that the first 4 characters of *ShortNickName must match the first 4 characters of *Manufacturer, at least for color printers that have ICC profiles in the Windows 95 system. Finally, because some print managers use *ModelName instead of *ShortNickName for ICC profile matching, and the builder of the PPD file doesn't know which print manager will be used, *ModelName should match *ShortNickName to ensure* *consistent ICC profile matching. See the Microsoft document "Image Color Matching and Printer Drivers in Windows 95" for details on how color profiles are matched to devices and drivers and how the* ic.model *profile tag is generated.*

5.4 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, memory modules, and so on. The PPD specification provides a way to describe these accessories, to 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 option that is not installed and get errors or unexpected results.

Syntax and Use

The *OpenGroup structure keyword (described in section 5.2) 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 InstallableOptions 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 keyword initially denotes the state of the accessory in the minimal configuration; that is, whether it is installed or not when the device leaves the factory. If the state of the accessory can be determined by querying the PostScript interpreter, there may also be query code, which a print manager operating in a bidirectional environment could use to update its information about the device's configuration.

Because there is no need for them to have meaningful names, the main keywords used within the *OpenUI/*CloseUI entries consist of the generic string *Option followed by an integer; for example, *Option1, *Option2. Each installable option (each *OpenUI entry) must have a unique main keyword 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 statement tells a print manager that if *Option1 is False (the envelope feeder is not installed), then the Envelope option of the *InputSlot keyword is not available for selection by the user.

It is also legal to have a named keyword within the InstallableOptions group. This might be done for a keyword whose effect is important enough or complex enough that a print manager might want to generate a separate configuration panel for that keyword. A named keyword might also be used where that keyword needs to be matched to other keywords by some method other than *UIConstraints. In either case, the print manager needs to be able to recognize that keyword. For an example, see the discussion of *InstalledMemory in section 5.6 and later in this section. In general, the use of the generic keywords is recommended, to discourage special casing by print managers, keyword proliferation, and additional documentation.

A print manager can use the InstallableOptions group in at least 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 previous examples, but PickOne entries are equally legal. For example, the following entry provides a short list of mutually exclusive choices (the user can install 2MB or 4MB of memory, but not both at once). This is also an example of using a named keyword instead of a generic main keyword:

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

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

This *UIConstraints entry tells a print manager that if None has been selected for *InstalledMemory, then neither the Medium nor Dark options of the *Smoothing keyword are available, and if 2Meg has been selected for *InstalledMemory, 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 typically null quotes because no code is invoked during configuration; the print manager is simply recording information either from the user or from the *Default statements. However, in some cases, there may be actual PostScript code between the quotes, perhaps to perform some type of job setup related to the device's configuration. In that case, the *OpenUI entry must also contain an *OrderDependency statement, so that the print manager knows where to insert the code in the job stream.

5.5 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 print 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 statement will 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 statement 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.
- TrueImage—This device contains a TrueImage rasterizer, which accepts the TrueImage version of a TrueType font.

***?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.

***1284Modes** *channelOption*: *mode...*

The StringValue of this keyword describes the level of compliance of each communication channel with the IEEE 1284-1994 specification, *IEEE Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers*. It tells which communication modes are supported, on a per-channel basis. For example:

```
*1284Modes Parallel: Compat Nibble
```

The *channelOption* must match one of the channel names listed under *Source. (See section 5.21 for information on *Source.) More than one *mode* may be listed in the value field, separated by white space, meaning multiple modes are supported on that channel. The *mode* definitions are taken from the 1284-1994 specification, and are as follows:

- Compat—Compatibility mode (unidirectional host-to-peripheral parallel communication). Many older devices support only this mode.
- Nibble—4-bit reverse (peripheral-to-host) communication
- Byte—8-bit reverse (peripheral-to-host) communication
- ECP—Extended Capabilities Port (8-bit bidirectional)
- EPP—Enhanced Parallel Port (8-bit bidirectional)

Currently, the only channel described by the 1284-1994 specification is Parallel, but other channels may add support in the future, so other channel options are permitted. There may also be variations of Parallel on a device, such as ParallelB for a second parallel channel.

***1284DeviceID:** *"text"*

For a device that is compliant with the IEEE 1284-1994 specification mentioned under *1284Modes, the QuotedValue of this keyword provides the Device ID string returned by the device. For example:

```
*1284DeviceID: "MFG:Acme;MODEL:SuperSpiffy;COMMAND SET: POSTSCRIPT,PJL,PCL”
```

If the device does not return a valid string in response to a request for Device ID, this statement will be omitted from the PPD file.

5.6 System Management

† ***PatchFile:** *“invocation”*

This represents a (perhaps large) PostScript language sequence that is a downloadable patch to ROM code, which must be 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 certain 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.
- Compare the **product**, **version**, and **revision** strings on the printer to the values of the *Product and *PSVersion statements 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.

Please see the comments on daggered keywords at the end of section 4.1 for a list of the additional responsibilities of the builder of the PPD file and of print manager authors.

---

† This keyword requires the *Password value to be supplied in front of the invocation.

***?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 to apply a bug fix or to set up an initial state for a job, but it does not require a password and is not downloaded outside the server loop. Its code 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 for a device may be numbered.

***FreeVM:** *"int"*

This keyword gives the maximum amount of memory available for use by a PostScript language job in the product's minimal memory configuration. The QuotedValue is the integer returned by the PostScript language sequence

```
vmstatus exch sub == pop
```

or, on a Level 2 device,

```
2 vmreclaim vmstatus exch sub == pop
```

executed immediately after the device is first powered on.

For a print manager, *FreeVM does not necessarily reflect the current amount of VM available on the device, since either more memory may have been added or VM may have been used up by downloaded fonts or other resources. It should be regarded as a maximum limit of free VM in the minimal memory configuration, rather than as a measure of current availability. Historically, *FreeVM has been used by print managers to determine which of several devices has more memory built into it. Because *FreeVM is generated using the product's minimal memory configuration, it is of limited use on devices that support additional memory modules. In that case, the combination of *InstalledMemory and *VMOption is more useful for determining how much memory might be available.

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

This keyword provides potential values of the *FreeVM keyword with various optional memory (VM) configurations installed. The values are obtained by inserting each additional memory module, one at a time, and recording the value returned by the code fragment listed in the description of *FreeVM.

The *vmOption* None denotes the basic, standard memory configuration, with no additional memory upgrades. The other *vmOptions* must match the *vmOptions* in the *InstalledMemory entry, and are generally of the form 2Meg or 2MB, denoting the size of the total installed memory. The value, although in the form of an InvocationValue, must be an integer.

A PPD file with a *VMOption entry must also have a *FreeVM statement. At least one *VMOption statement must contain the same value as the *FreeVM statement.

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"
```

*VMOption is used with *InstalledMemory to determine how much memory is installed and how much VM is available as a result. See the description of *InstalledMemory for an explanation.

Note *To builders of PPD files: PPD files with *FormatVersion statements greater than 4.1 should not have *UIConstraints between an installable memory module and *VMOption. This method is obsolete. Instead, there should be *InstalledMemory entries in the InstallableOptions group for each additional amount of memory. *InstalledMemory is automatically linked to *VMOption by the use of identical option keywords, so there is no need for *UIConstraints on *VMOption.*

***InstalledMemory** *vmOption*: *"invocation"* UI

***DefaultInstalledMemory**: *vmOption* | Unknown

***?InstalledMemory:** *"query"* (returns: *vmOption* | Unknown)

*InstalledMemory is used to link the amount of physical memory in the device with the amount of available VM. The string *vmOption* must match a valid *vmOption* listed under *VMOption. *DefaultInstalledMemory indicates the default amount of memory installed. *?InstalledMemory returns a *vmOption* that corresponds to the current memory configuration. Note that this is **not** the amount of free VM, but an indicator of the physical memory installed.

This entry would typically appear in the InstallableOptions group. If the *?InstalledMemory query is present and a bidirectional communication channel is available, the print manager can use the *?InstalledMemory query to ask the device how much memory is installed. If a bidirectional communication channel is not available, the print manager can use *InstalledMemory to ask the user how much memory is installed. (See section 5.4 for a description of the InstallableOptions group.) The print manager would then match the user's choice of an option for *InstalledMemory (or the string returned by *?InstalledMemory) to the same option of *VMOption to find out how much VM is available.

For example, in a unidirectional environment, the print manager could read the following entry and add it to a configuration panel, asking the user to select which memory module had been installed in their printer:

```
*OpenGroup: InstallableOptions
*OpenUI *InstalledMemory: PickOne
*DefaultInstalledMemory: 2MB
*InstalledMemory 2MB/Standard: " "
*InstalledMemory 3MB/3 MB Upgrade: " "
*InstalledMemory 4MB/4 MB Upgrade: " "
*CloseUI: *InstalledMemory
*CloseGroup: InstallableOptions
```

The print manager could then take the user's selection (let's say it was 4MB) and search the following statements to find out how much VM would be provided by the 4MB option:

```
*VMOption 2MB: "1234567"
*VMOption 3MB: "2345678"
*VMOption 4MB: "3456789"
```

This provides a rough method of obtaining an updated value for *FreeVM. While this value still may not reflect the true amount of VM available, due to resource downloading, in a communications environment where the print manager cannot query the printer for the actual amount of VM available, this provides something closer to the truth.

In most cases, the invocation value for *InstalledMemory will be null; that is, there will be no code between the quotes (like most values in the InstallableOptions group). However, in some cases, the invocation value might contain code, in which case the print manager should treat it as normal invocation code. If there is actual code in the quotes, the *InstalledMemory entry must have an *OrderDependency statement specifying where the code should be emitted. The *InstalledMemory keyword can also occur outside the InstallableOptions group.

† ***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. Please see the comments on daggered keywords at the end of section 4.1 for a list of the additional responsibilities of the builder of the PPD file and of print manager authors.

***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.

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

***DefaultExitJamRecovery:** True | False | Unknown

***?ExitJamRecovery:** *"query"* (returns: True | False | Unknown)

This keyword provides the code to invoke the "exit jam recovery" feature. If True, pages that jam in the exit path are reprinted. If False, jammed pages are not reprinted, which may result in a performance improvement because more overlapping of page processing is possible. *DefaultExitJamRecovery denotes the default state. *?ExitJamRecovery returns the current state.

† ***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 used to apply bug fixes 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. Please see the comments on daggered keywords at the end of section 4.1 for a list of the additional responsibilities of the builder of the PPD file and of print manager authors.

† This keyword requires the *Password value to be supplied in front of the invocation.

***SuggestedJobTimeout:** "*int*"

This QuotedValue provides the time, in seconds, that the device manufacturer suggests for the value of the user parameter **JobTimeout** (Level 2) or the argument to **setjobtimeout** (Level 1). This value may be the default value set in the device at the factory, or it may be an alternative to the factory-set value, provided for performance or other reasons. This keyword is intended for print managers that allow the user to change the job timeout value; it provides an initial value for display to the user.

***SuggestedManualfFeedTimeout:** "*int*"

This QuotedValue provides the time, in seconds, that the device manufacturer suggests for the value of the page device parameter **ManualFeedTimeout** (Level 2) or the argument to **setmanualfeedtimeout** (Level 1). This value may be the default value set in the device at the factory, or it may be an alternative to the factory-set value, provided for performance or other reasons. This keyword is intended for print managers that allow the user to change the manualfeed timeout value; it provides an initial value for display to the user.

***SuggestedWaitTimeout:** "*int*"

This QuotedValue provides the time, in seconds, that the device manufacturer suggests for the value of the user parameter **WaitTimeout** (Level 2) or the argument to **setwaittimeout** (Level 1). This value may be the default value set in the device at the factory, or it may be an alternative to the factory-set value, provided for performance or other reasons.This keyword is intended for print managers that allow the user to change the wait timeout value; it provides an initial value for display to the user.

***PrintPSErrors:** True | False

This StringValue indicates to a print manager whether or not the device manufacturer thinks that PostScript interpreter error information should be printed on the device. Printing interpreter error information is appropriate on some devices, but not on others. If True, the device manufacturer suggests that printing interpreter error information is appropriate for this device. A print manager may, of course, let the user override this suggested setting; this keyword is intended to provide default behavior for a print manager, and a value for initial display to the user if the behavior is to be changed.

Note *To builders of PPD files: As a rule of thumb,* *PrintPSErrors *is usually set to* True *for cut-sheet devices and* False *for roll-fed devices.*

***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 problems, the value of *DeviceAdjustMatrix is the identity matrix [1 0 0 1 0 0], and the entire statement will be 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.*

5.7 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 protocols 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 Technical Note #5009, *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 statement 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 Technical Note #5009, *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, separated by white space.

For each *emulatorOption* listed under *Emulators, there must also be corresponding main keywords that 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. This odd syntax allows the values to be QuotedValues.

For example:

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

An *emulatorOption* must appear in a *Emulators statement before the corresponding *StartEmulator_ and *StopEmulator_ keywords appear.

The currently registered option keywords for *emulatorOption* 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 "
*End
```

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 "
*End
*StopEmulator_hplj: "<1B7F>0"
```

5.8 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 that 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.

If a feature can be selected either through the PostScript interpreter or through JCL, the device manufacturer should decide which method is preferred and should use only one method in the PPD file. Although it is legal to include both the PostScript and JCL methods of invoking a feature in the PPD file, it is not recommended for the following reasons:

- The result is undefined. For example, if a user sets the resolution using *JCLResolution and later sets the resolution differently using *Resolution, the resolution result will depend on the order in which the print manager performs the operations and on how the two methods interact in the device.
- A print manager, parsing blindly for *JCLOpenUI and *OpenUI, may offer the user two methods of changing the resolution on the same print panel, which would be confusing.

***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 5.2 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"*

DefaultJCLFrameBufferSize:* *frameBufferOption* | Unknown

***?JCLFrameBufferSize**: *"query"* (returns: *frameBufferOption* | Unknown)

*JCLFrameBufferSize 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 may contain out-of-range byte codes in hexadecimal strings, which the print manager must translate before emitting to 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 5.13 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
```

*DefaultJCLFrameBufferSize indicates the default frame buffer size set by a JCL command. *?JCLFrameBufferSize returns a string denoting the current frame buffer size set by a JCL command. If it is never possible to determine the frame buffer size, the *?JCLFrameBufferSize query will be omitted.

***JCLResolution** *resolutionOption*: *"JCL"'*

***DefaultJCLResolution**: *resolutionOption* | Unknown

***?JCLResolution**: *"query"* (returns: *resolutionOption* | Unknown)

*JCLResolution provides the JCL code to change the resolution. There is one statement for each resolution supported by the device. For a complete explanation of *resolutionOption* and its possible values, see the description of *DefaultResolution in section 5.9.

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.*

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
```

*DefaultJCLResolution indicates the default resolution set by a JCL command. *?JCLResolution returns a string denoting the device resolution set by a JCL command. If it is never possible to determine the resolution, this query will be omitted.

5.9 Resolution and Appearance Control

This section contains keywords that control the resolution and related appearance characteristics of the device.

***DefaultResolution:** *resolutionOption* | Unknown

This statement provides the default resolution of the device. The resolution is measured in dots (spots) per linear inch, in both *x* and *y* dimensions, 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 at power-up. If *DefaultResolution is part of an entry, the value of *resolutionOption* appearing here must be a valid resolution listed under *SetResolution or *Resolution.

If the format of *resolutionOption* is 300x300dpi, this signifies that 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.” This format should be used only for a device that supports anamorphic resolution.

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 format 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.

Note *Builders of PPD files: If the device has only one resolution, *DefaultResolution may appear by itself, without *Resolution, *SetResolution, or any *OpenUI/*CloseUI bracketing. See section 3.2 and section 4.5 for information on stand-alone default keywords.*

***Resolution** *resolutionOption*: *"invocation"* UI

For devices that support resolution changes from within a PostScript language job, this keyword will provide the proper InvocationValue for each resolution supported by the device. There can be several of these statements, if the PostScript device supports multiple selectable resolutions. The string *resolutionOption* is of the form specified in the *DefaultResolution statement. Print managers need to ensure that any resolution changes occur before the page size is selected.

Note *To builders of PPD files: *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 keyword will provide the proper InvocationValue for each resolution supported by the device. There can be several of these statements, 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. Please see the comments on daggered keywords at the end of section 4.1 for a list of the additional responsibilities of the builder of the PPD file and of print manager authors.

Note *To builders of PPD files: *SetResolution 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.*

† This keyword requires the *Password value to be supplied in front of the invocation.

***?Resolution:** *"query"* (returns: *resolutionOption* | Unknown)

This query returns a string denoting the current resolution of the device. The returned value will be a string in the format specified by *DefaultResolution, including the string "dpi", or it will be Unknown. The resolution returned must be a valid resolution listed under *SetResolution or *Resolution, if those entries are present. Upon device power-up, downloading the *?Resolution code to the device should return the value of *DefaultResolution.

***Smoothing** *smoothOption: "invocation"* UI

***DefaultSmoothing:** *smoothOption* | Unknown

***?Smoothing:** *"query"* (returns: *smoothOption* | Unknown)

*Smoothing provides the InvocationValues to invoke various levels of "smoothing" the edges of text and graphics after they have been rendered by the device. This 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 denotes the default state of the smoothing mechanism. *?Smoothing returns a string that denotes the current state of the smoothing mechanism.

***BitsPerPixel** *depthOption*: *"invocation"*

UI

***DefaultBitsPerPixel:** *depthOption* | Unknown

***?BitsPerPixel:** *"query"* (returns: *depthOption* | Unknown)

*BitsPerPixel provides the InvocationValues to select various gray-scale levels or color depths. *depthOption* is a string that denotes the number of bits per pixel that should be used to represent a color when rendering the job on the device. The currently registered values for *depthOption* are:

- None, Off, False—Used to represent the lowest number of bits per pixel, which is typically 1.
- On, True—Used with Off and False respectively when this feature has only two states. Represents the highest number of bits per pixel available.
- 2—Use 2 bits per pixel.
- 4—Use 4 bits per pixel.
- 8—Use 8 bits per pixel.

*DefaultBitsPerPixel denotes the default state of color depth. *?BitsPerPixel returns a string that denotes the current color depth.

5.10 Gray Levels and Halftoning

***AccurateScreensSupport:** True | False

This StringValue 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*.

***ContoneOnly** True | False

This StringValue indicates the continuous tone capabilities of the device. This keyword only appears if the device can reproduce color (including grayscale) as continuous tones. True means the device can reproduce color and grayscale only as continuous tones; it cannot halftone. False means the device can reproduce color and grayscale either as continuous tones or halftones. The absence of this keyword means that the device cannot produce continuous tones at all, but can only produce halftones. An application can use this information to decide whether or not to download special halftone dictionaries, or whether to even offer this capability to the user. There is no point in downloading halftone dictionaries if *ContoneOnly is True.

***DefaultHalftoneType:** *int*

This StringValue is the integer value of **HalftoneType** in the default **Halftone** dictionary. This keyword applies only to Level 2 devices. This keyword is present only if the device is capable of halftoning and if the default state of the device is to produce halftones rather than continuous tone color. (*ContoneOnly:True and *DefaultHalftoneType may not appear in the same file.)

*DefaultHalftoneType provides a hint to the print manager about the accuracy of the values for *ScreenFreq, *ScreenAngle, and *DefaultScreenProc. See their descriptions for further information.

***ScreenAngle:** *"real"*

***ScreenFreq:** *"real"*

***DefaultScreenProc:** *spotOption*

These keywords provide the default halftone screen angle, frequency, and spot function, respectively. On Level 1 devices, these values are the *angle, frequency,* and *proc* arguments returned by the **currentscreen** operator after powering on the device. On Level 2 devices, only a type 1 halftone dictionary can be easily represented by these keywords. Therefore, if the value of *DefaultHalftoneType is 1, or if *DefaultHalftoneType is not present, these values are the **Frequency, Angle,** and **SpotFunction** entries in the default **Halftone** dictionary. If the value of *DefaultHalftoneType is anything other than 1, the values of *ScreenAngle, *ScreenFreq, and *DefaultScreenProc may be meaningless and application authors may not want to rely on them for anything important.

Note *To builders of PPD files: Although these are not required keywords, *ScreenFreq, *ScreenAngle, and *DefaultScreenProc should be present even for contone-only devices, because many applications have come to depend on their presence, even though their values may be useless on a particular device. You should assume that some applications will execute **setscreen** with the values provided by *ScreenFreq, *ScreenAngle, and *DefaultScreenProc. To prepare for this, if the value of *DefaultHalftoneType is anything other than 1, you should put reasonable values for angle, frequency, and spotOption in the keywords above. If you don't know what values to use, try 45 for angle, 60 for frequency, and Dot for spotOption, and make sure that *ScreenProc Dot is defined in the PPD file.*

Note *Some older PPD files for Level 2 devices may need to have these values corrected. Prior to the 4.3 version of this specification, there was no requirement that these values represent the **Frequency, Angle,** and **SpotFunction** entries in the default **Halftone** dictionary in a Level 2 device, and if **sethalftone** was use to set these values, the **currentscreen** operator used to build the PPD files would not have returned the correct values.*

*ScreenAngle and *ScreenFreq are QuotedValues, and *DefaultScreenProc is a StringValue. For *DefaultScreenProc, the *spotOption* must correspond to one of the options listed under *ScreenProc.

***ResScreenFreq** *resolutionOption*: *"real"*

***ResScreenAngle** *resolutionOption*: *"real"*

On devices with user-settable resolution, the halftone screen frequency and angle may be changed by the device when the resolution is changed by the user. These keywords provide the halftone screen frequency and angle that is applied by the device for each settable resolution. The option must be a valid *resolutionOption* listed under *Resolution, *SetResolution, or *JCLResolution in the PPD file for this device. There should be one *ResScreenFreq and *ResScreenAngle statement for each settable resolution. See section 5.9 for an explanation of the format of *resolutionOption*.

On Level 2 devices, only a type 1 halftone dictionary can be easily represented by these keywords. Therefore, for a Level 2 device, if the value of *DefaultHalftoneType is anything other than 1, these keywords should be omitted from the PPD file.

***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.

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 that 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 } "*

***DefaultTransfer:** Null | Factory

*Transfer 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 *To print manager authors: 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.*

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 option 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.

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.

On monochrome devices, *DefaultTransfer provides 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.

5.11 Color Adjustment

This section contains keywords used to adjust colors on color devices.

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

***DefaultBlackSubstitution:** True | False | Unknown

***?BlackSubstitution:** *"query"* (returns: True | False | Unknown)

*BlackSubstitution 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 a better black. *DefaultBlackSubstitution denotes the default state of the black substitution feature. *?BlackSubstitution returns True if black substitution is currently invoked and False if it is not.

***ColorModel** *colormodelOption*: *"invocation"*

UI

***DefaultColorModel:** *colormodelOption* | Unknown

***?ColorModel:** *"query"* (returns: *colormodelOption* | Unknown)

*ColorModel 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 denotes the default native color model of the device. *?ColorModel returns the current native color model. 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 built-in color rendering dictionaries (CRDs) to calibrate the device colors for different rendering intents, different types of paper, different halftone screens, or for other purposes. This keyword lists the CRDs that are built into the device and provides the InvocationValue code to invoke each CRD referred to by *dictOption*. There will be one instance of this keyword for each built-in color rendering dictionary. The invocation code sets up the named CRD to be the current CRD, which will affect all imaging done after this code appears in the output file.

In devices with interpreter versions below 2015, CRD names were arbitrary strings. In devices with interpreter versions of 2015 and later, CRDs must be named according to Adobe's CRD naming conventions. For more information on naming and using color rendering dictionaries, see section 5.4, *CRD Selection Based on Rendering Intent*, in the *PostScript Language Reference Manual Supplement for Version 2015*, available from the Adobe Developers Association.

The *dictOption* name will be a concatenation of rendering intent, device setup, and halftone names, separated by dots:

```
renderingintent.devicesetup.halftone
```

The name will also typically have a translation string. For example:

```
*ColorRenderDict Saturated.6x6Transparency.Dot/Saturated Color, 600x600 dpi, Transparency, Dot:
  "/Saturated.6x6Transparency.Dot /ColorRendering findresource setcolorrendering"
*End
```

An application might provide the list of built-in CRDs to the user for selection, so a meaningful translation string is important. See *RenderingIntent, *PageDeviceName, and *HalftoneName for more information on the components of a CRD name.

In addition, if a user wants to supply new CRDs that are not built into the device, new instances of this keyword can be added to a local customization PPD file for a given device. This is rarely used because it greatly increases the size of the PPD file. For a new CRD in a local customization file, the *ColorRenderDict code would have to create the CRD, fill it with the appropriate values for color calibration, and invoke the CRD with **setcolorrendering**. If the CRD is to be made available for future use as a resource, the appropriate **defineresource** code and any other necessary procedures must also be included.

Note *To application developers: There is a close relationship between the current device setup (page device), the current halftone dictionary, and the appropriate CRD. The *ColorRenderDict code does not take these things into account. Application developers who wish to use *ColorRenderDict should ensure that the appropriate halftone and device setup are invoked correctly before executing the *ColorRenderDict code, or the results will be unpredictable. Better yet, applications should use* ***findcolorrendering*** *to find and set up the appropriate CRD.*

***RenderingIntent:** *string*

This keyword provides the list of rendering intents supported by the device. At minimum, all of the intents named in the built-in CRDs must appear here, as StringValues. These should correspond to the first component of each *dictOption* listed under *ColorRenderDict. The manufacturer may also choose to list any additional built-in intents.

Here is an example, listing the four standard rendering intents:

```
*RenderingIntent: AbsoluteColorimetric
*RenderingIntent: RelativeColorimetric
*RenderingIntent: Saturation
*RenderingIntent: Perceptual
```

For more information on the name components of color rendering dictionaries, see section 5.4, *CRD Selection Based on Rendering Intent*, in the *PostScript Language Reference Manual Supplement for Version 2015*, available from the Adobe Developers Association.

***PageDeviceName:** *string*

This keyword provides the list of device setups (page device names) supported by the device. At minimum, all of the device setups named in the built-in CRDs must appear here, as StringValues. These should correspond to the second component of each *dictOption* listed under *ColorRenderDict. The manufacturer may also choose to list any additional built-in page device names. These page device names must correspond either to instances of the **setpagedevice** key **PageDeviceName**, or to the list of names returned by the built-in procedure **GetPageDeviceName**, or the list may include all of the above names.

For example:

```
*PageDeviceName: 6x6Transparency
*PageDeviceName: 3x3Transparency
*PageDeviceName: Paper
```

For more information on the name components of color rendering dictionaries, see section 5.4, *CRD Selection Based on Rendering Intent*, in the *PostScript Language Reference Manual Supplement for Version 2015*, available from the Adobe Developers Association.

***HalftoneName:** *string*

This keyword provides the list of halftone names supported by the device. At minimum, all of the halftones named in the built-in CRDs must appear here, as StringValues. These should correspond to the third component of each *dictOption* listed under *ColorRenderDict. The manufacturer may also choose to list any additional built-in halftone names. These halftone names must correspond either to instances of the **Halftone** dictionary key **HalftoneName**, or to the list of names returned by the built-in procedure **GetHalftoneName**, or the list may include all of the above names.

For example:

```
*HalftoneName: ScatterDot
*HalftoneName: QuadDot
```

For more information on the name components of color rendering dictionaries, see section 5.4, *CRD Selection Based on Rendering Intent*, in the *PostScript Language Reference Manual Supplement for Version 2015*, available from the Adobe Developers Association.

Note **RenderingIntent, *PageDeviceName, and *HalftoneName may be useful to an application that wishes to download a new CRD without immediately invoking it (*ColorRenderDict immediately invokes the new CRD). In that case, the application or user must provide the contents of the new CRD and the code to down-*

load it to the device, name it properly, define it as a resource, and perform any other necessary procedures. These keywords provide components for the new CRD name.

5.12 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 the user wants 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, print managers should use the *PageSize keyword to request media.

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 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, before any marks are made on the page, and outside of any page-level* ***save****.*

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 manager 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 language interpreter is described by the *ImageableArea keyword.

5.13 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 statements *PageSize Letter, *PaperDimension Letter, and *ImageableArea Letter all address different characteristics of a letter-size page.

Tables of the currently registered media option keywords, sorted by both name and size, can be found in *Appendix B: Registered mediaOption Keywords.* The media option names in those tables can be substituted for any occurrence of the placeholder word *mediaOption* anywhere in this specification.

Additional media option keywords can be added to the list of registered option keywords at any time. To ensure that the set of device features is not artificially limited, a print manager should parse the PPD file for the complete list of option keywords in a main keyword entry, rather than parsing for specific option keywords. See section 5.1 for more information about the extensibility of option keywords and the rules for creating new ones.

With closely related statements, such as the media handling keywords, it is impossible to predict which statement a print manager will read to get the translation string for an option keyword. For continuity of results, if a *mediaOption* of one main keyword has a translation string, and that *mediaOption* is used with multiple main keywords and has the same semantics across those keywords, then the translation string should be on every occurrence of the *mediaOption* and should be identical across occurrences. For example, if the *PageSize statement for Letter uses a translation string Portrait Letter, then the *PageRegion, *PaperDimension, and *ImageableArea statements for Letter should all use the same translation string Portrait Letter.

5.14 Media Selection

The keywords in this section allow the user to control the selection of media by specifying characteristics such as page size, input slot, media type, media color, and other attributes.

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

UI

***DefaultInputSlot:** *inputSlotOption* | Unknown

***?InputSlot:** *"query"* (returns: *inputSlotOption* | Unknown)

*InputSlot provides the InvocationValue to select media by specifying the name of the input tray in which the media is located, rather than the page size or other characteristics of the media. For example, the media can be selected by specifying 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. There will be one statement for each software-selectable input slot. *DefaultInputSlot provides the name of the default input slot. *?InputSlot returns the name of the current input slot.

Any arbitrary strings that appropriately describe the devices's input slots are valid *inputSlotOption*s. The following list documents commonly used *inputSlotOption*s. For Windows print managers, Microsoft has defined C language constructs called #defines, which are used to match Windows application requests for input slots to the *inputSlotOption* names in PPD files. If one exists, the appropriate #define is listed in the *inputSlotOption* description, for use by print managers. Builders of PPD files are encouraged to use these standard *inputSlotOption* names so that Windows print managers can correctly match application requests to input slots:

- Lower—This is used for any tray which has no particular distinguishing feature other than it is lower than another tray similar to it.
  #define: DMBIN_LOWER
- Middle—This designates a tray that is between other trays. See Lower.
  #define: DMBIN_MIDDLE
- Upper—This designates a tray that is above other trays. See Lower.
  #define: DMBIN_UPPER
- Rear—This designates a tray at the rear of the device.
- Envelope—This denotes an envelope tray. #define: DMBIN_ENVELOPE
- Cassette—This keyword can be used where Upper, Middle, and Lower make little sense (for example, 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. #define: DMBIN_CASSETTE
- 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.
  #define: DMBIN_LARGECAPACITY

- 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. #define: DMBIN_SMALLFMT

- 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).#define: DMBIN_LARGEFMT

Option keywords may also combine other attributes such as *MediaType or *ManualFeed with *InputSlot. For example:

```
*InputSlot ManualPaper: "code"
*InputSlot ManualTransparency: "code"
```

The code fragments would select the manual feed slot and set up the printer (perhaps adjusting color densities) to print on paper or transparency, respectively.

Note *To builders of PPD files: If features are combined into an option keyword as shown above, the relevant feature keyword should usually be omitted, to avoid presenting the user with two different ways to choose a feature. In this example, the *ManualFeed and *MediaType entries should be omitted, since the use of *InputSlot allows selection of those features. Likewise, if *InputSlot Manual is present, the *ManualFeed entry should be omitted, to avoid providing two methods of invoking the manual feed slot. Use common sense and test the PPD file with print managers to make sure a feature is not presented in multiple ways, confusing the user.*

Note *To builders of PPD files: The *InputSlot entry is not required. However, even if there is only one input slot, a minimal *InputSlot entry is usually included. This allows the manufacturer to dictate the slot name (as an option keyword or translation string) for a print manager to display, rather than using the print manager's default choice of a slot name. Cassette is the most commonly used name for single-slot or roll-fed devices. For example:*

```
*OpenUI *InputSlot: PickOne
*OrderDependency: 20 AnySetup *InputSlot
*DefaultInputSlot: Cassette
*InputSlot Cassette: ""
*CloseUI: *InputSlot
```

*There is no need to include the *?InputSlot query, since it provides no useful information and increases the size of the PPD file, but it can be included, with code that simply flushes back the string Cassette. For example:*

```
*?InputSlot: "save (Cassette) = flush restore"
```

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

***DefaultManualFeed:** True | False | Unknown

***?ManualFeed:** *"query"* (returns: True | False | Unknown)

*ManualFeed provides the InvocationValue to turn manual feed on (True) and off (False). *DefaultManualFeed denotes the default state of the manualfeed mechanism. *?ManualFeed returns the current state of the manual feed mechanism.

Some manufacturers prefer to handle the manual feed slot as a regular input slot, naming one of the *InputSlot options Manual or ManualFeed and invoking manual feeding in the code for that option. In this case, *ManualFeed should not be present, as that would cause a print manager to offer the user two different methods of choosing manual feed, which would be confusing.

Note *To builders of PPD files: The existence of *ManualFeed as a separate feature is an historic anomaly. Adobe recommends that you omit *ManualFeed and instead include *InputSlot Manual or *InputSlot ManualFeed. This provides a cleaner interface for print managers, which usually regard the manual feed slot as just another input slot. It also means that you don't have to write *UIConstraints between *ManualFeed and every *InputSlot option, thus reducing both the size of the PPD file and the time it takes to build it.*

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

***DefaultPageSize:** *mediaOption* | Unknown

***?PageSize:** *"query"* (returns: *mediaOption* | Unknown)

Required. *PageSize provides the InvocationValue to invoke supported page sizes. *DefaultPageSize indicates the default page size set by the device when it is first powered up. Since there can be only one default page size, this value should be the same as the value of *DefaultPageRegion, *DefaultImageableArea, and *DefaultPaperDimension. *?PageSize returns the media option corresponding to the current page size, and is not required if it is not possible to write such a query.

The *PageSize invocations 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 by a print manager 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).

Note *To print manager authors: An invocation string supplied by* *PageSize *will usually override an invocation string supplied by* *PageRegion*. Therefore, if, for some reason, both a* *PageRegion *invocation and a* *PageSize *invocation for a single page are going into the output file, the* *PageRegion *invocation must come after the* *PageSize *invocation to achieve the expected results.*

Note *To builders of PPD files: In a PPD file for an imagesetter, the invocation strings for* *PageSize *and* *PageRegion *are usually identical. On devices that support multiple page sizes, the value of* *DefaultPageSize *will often be* Unknown, *as it may be impossible to predict which media tray will be inserted or designated as the default tray. Also, read the end of section 5.13 for a discussion of translation strings on media option keywords.*

Currently registered values for *mediaOption* may be found in *Appendix B: Registered mediaOption Keywords*

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

***DefaultPageRegion:** *mediaOption* | Unknown

Required. The InvocationValues of *PageRegion 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 indicates the default imageable area (in terms of media options) for the device when powered on. Since there can be only one default page size, this value should be the same as the value of *DefaultPageSize, *DefaultImageableArea, and *DefaultPaperDimension.

Note *To print manager authors:* *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 5.17.*

Note *To builders of PPD files: In a PPD file for an imagesetter, the invocation strings for* *PageSize *and* *PageRegion *are usually identical. On devices that support multiple page sizes, the value of* *DefaultPageRegion *will often be* Unknown, *as it may be impossible to predict which media tray will be inserted or designated as the default tray. Also, read the end of section 5.13 for a discussion of translation strings on media option keywords.*

***MediaType** *typeOption*: “*invocation*” UI

***DefaultMediaType:** *typeOption* | Unknown

***?MediaType:** “*query*” (returns: *typeOption* | Unknown)

*MediaType provides the InvocationValue to select media by some characteristic other than size (or in addition to size). The *typeOptions* 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 provides the default media type. *?MediaType returns the current media type.

***MediaColor** *colorOption*: “*invocation*” UI

***DefaultMediaColor:** *colorOption* | Unknown

***?MediaColor:** “*query*” (returns: *colorOption* | Unknown)

*MediaColor provides InvocationValues to select media by color. The *colorOption*s are product-dependent strings that describe the available colors of media, such as Blue and Buff. This method usually requires prior device setup, so that the device knows how to access a certain color of media. *DefaultMediaColor provides the default media color. *?MediaColor returns the current media color.

***MediaWeight** *weightOption*: *“invocation”* UI

***DefaultMediaWeight:** *weightOption* | Unknown

***?MediaWeight:** *“query”* (returns: *weightOption* | Unknown)

*MediaWeight provides InvocationValues to select media by weight. The *weightOptions* 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 provides the default media weight. *?MediaWeight returns the current media weight.

5.15 Information About Media Sizes

The keywords in this section provide information about the media sizes that are available on the device. They do not invoke any device features.

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

***DefaultImageableArea:** *mediaOption* | Unknown

Required. *ImageableArea provides the bounding box of the imageable area for the page size named *mediaOption*. There will be one statement for each named page size supported by the device. *DefaultImageableArea provides the *mediaOption* name of the default imageable area. Since there can be only one default page size, this value should be the same as the value of *DefaultPageSize, *DefaultPageRegion, and *DefaultPaperDimension.

The bounding box value of *ImageableArea 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.

Note *To builders of PPD files: On devices that support multiple page sizes, the value of *DefaultImageableArea will often be Unknown, as it may be impossible to predict which media tray will be inserted or designated as the default tray. Also, read the end of section 5.13 for a discussion of translation strings on media option keywords.*

***?ImageableArea:** *"query"* (returns: *"ll_x ll_y ur_x ur_y"* | Unknown)

This query returns a string composed of 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 for a print manager to use the values supplied by the *ImageableArea statements, since they can be adjusted by hand for particular hardware constraints.

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

***DefaultPaperDimension:** *mediaOption* | Unknown

Required. The InvocationValue of *PaperDimension lists the physical dimensions of 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.

*DefaultPaperDimension provides the *mediaOption* name of the default physical media dimension. Since there can be only one default page size, this value should be the same as the value of *DefaultPageSize, *DefaultPageRegion, and *DefaultImageableArea.

Note *To builders of PPD files: On devices that support multiple page sizes, the value of *DefaultPaperDimension will often be Unknown, as it may be impossible to predict which media tray will be inserted or designated as the default tray. Also, read the end of section 5.13 for a discussion of translation strings on media option keywords.*

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

This keyword provides a StringValue that indicates, for each input slot, whether or not the *PageRegion invocation code must be sent with the *InputSlot invocation code when the user requests media from that input slot. For example, if the device cannot sense what page size is installed in a given input slot, 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 device cannot sense the page size in a given input slot, the *PageRegion code is required, and the value of *RequiresPageRegion will be True for that slot. The *PageRegion code may be required for any reason. If the *PageRegion code is not required for a given slot, then the value of *RequiresPageRegion for that slot 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 statement applies to all media sources on the device. For example:

```
*RequiresPageRegion All: False
```

This statement indicates that 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.

***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.

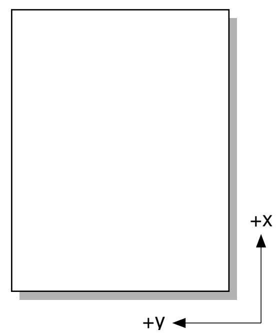
This keyword, whose StringValue is determined from knowledge of the printer's markings and instructions, provides a hint to a print manager about which way it should rotate and translate the page, for the printed output to be compatible with the page feeding instructions on the printer or in the printer's 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
  ```

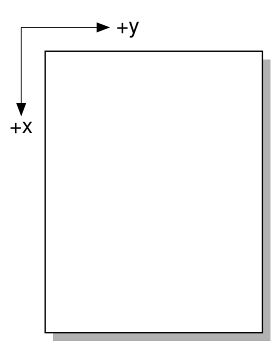
  where *pagewidth* is the width of the page in default user space. For example, on a letter-size page in portrait mode, after this transformation has been performed, the default user space would look like this:

- 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.*

5.16 Custom Page Sizes

Some devices support user-defined or *custom* page sizes by allowing the user to supply the page dimensions and other characteristics, rather than selecting from a list of pre-defined page sizes. The keywords in this section support that capability.

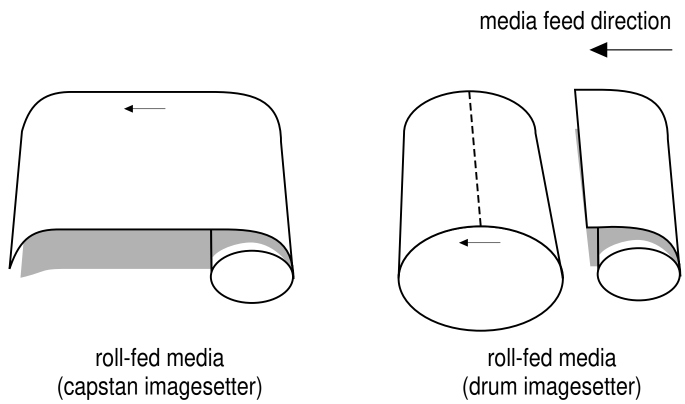
Custom page sizes are handled differently depending on whether the media is *roll-fed* or *cut-sheet*. Some devices accept both roll-fed and cut-sheet media. With roll-fed media, such as a roll of film or paper, the media is larger than the page size requested by the user. The requested page size is positioned somewhere on the larger physical media, and the imageable area may be assumed to be identical to the requested page size, which means the entire page area is imageable.

When using cut-sheet media, the user is expected to supply an individual sheet of the requested physical size, often in a tray that adjusts to different sizes. The page size requested by the user 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 described by the keyword *HWMargins.

Custom Page Size Parameters

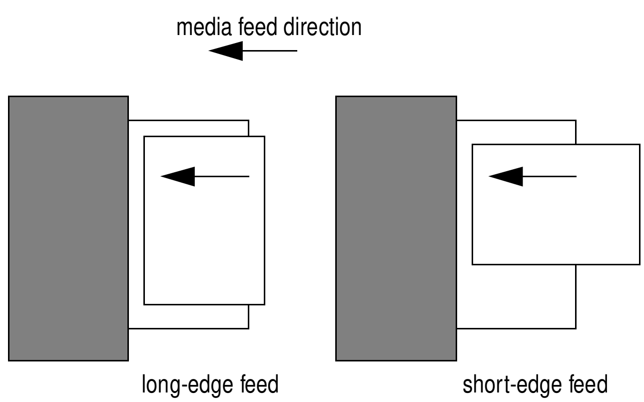
The location and orientation of the page image on the media and of the initial PostScript coordinate system depends on the combination of the custom page size parameters. Custom page size and orientation parameters are specified relative to the *media feed direction.* For roll-fed media, the media feed direction is parallel to the length of the roll of media, as shown in Figure 1:

Figure 1 *Media feed direction on roll-fed media*



For cut-sheet media, media feed direction is the direction in which media is fed into the device, as shown in Figure 2.

Figure 2 *Media feed direction on cut-sheet media*



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

- Width—This indicates the width of the page perpendicular to the direction of media feed, in PostScript default units.

- Height— This indicates the height of the page parallel to the direction of media feed, in PostScript default units.

- WidthOffset—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 when Orientation (defined below) is 0. A negative number indicates an offset in the direction of decreasing *y* in user space when Orientation is 0.

- HeightOffset—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 when Orientation (defined below) is 0. A negative number indicates an offset in the direction of decreasing *x* in user space when Orientation is 0.

- Orientation—This indicates the orientation of the image with respect to the media feed direction. Devices support a subset of four possible integer 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*, perpendicular to media feed direction. Orientations 1, 2, and 3 are rotated 90, 180, and 270 degrees (respectively) counterclockwise from orientation 0.

Note that Orientation does not provide a method of requesting a specific orientation of the physical page relative to the device. That is, Orientation does not request short-edge feed or long-edge feed of the physical page; it only requests a specific orientation of the image relative to the device. It is up to the user to feed the paper into the device in a direction that is compatible with the image they have requested.

The Orientation parameter can be used in several ways:

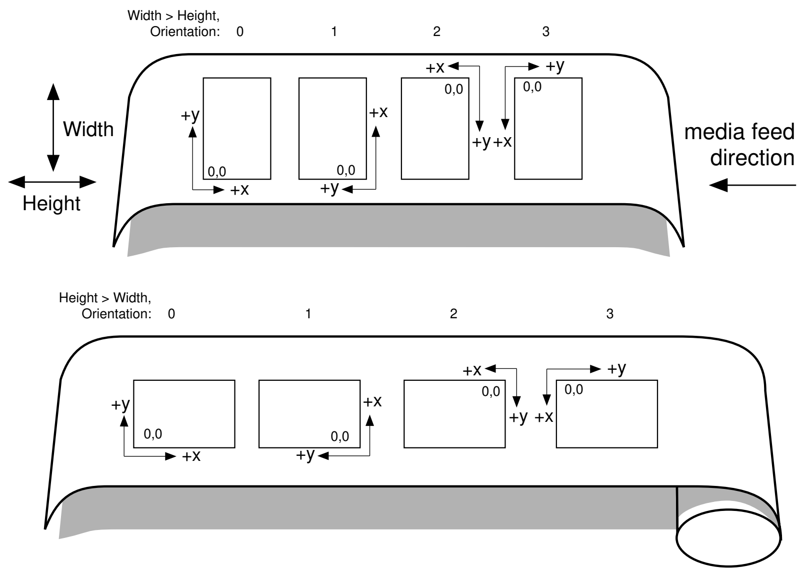
- If the device supports it, Orientation can be used in the *CustomPageSize code to request a specific image orientation from the device. Most roll-fed devices support this; most cut-sheet devices do not.

- For cut-sheet media, the primary use for the Orientation parameter is for the print manager to calculate the imageable area of the page, using the values provided by *HWMargins. See the description of *HWMargins for details.

- Orientation may also be used by the print manager in offering the user a pictorial representation of the image on the page.

Figure 3 shows the interaction between Width, Height and Orientation. Although this figure depicts roll-fed media, the same principles apply to cut-sheet media. 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 ($y < x$) coordinate system and two values of Orientation will produce a portrait ($x < y$) coordinate system.

Figure 3 *Interaction of Width, Height, and Orientation*



***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.

Because *CustomPageSize emits code, there must be a *NonUIOrderDependency statement for *CustomPageSize. There may be *NonUIConstraints between *CustomPageSize and *InputSlot or other features.

The *CustomPageSize code can be quite complex. See section 6.3 for several examples of complete custom page size entries on various types of devices.

Note *On a roll-fed device, the actual orientation of a page might not match the request, due to device configuration. For example, an imagesetter manufacturer might configure a product to conserve media by rotating a page automatically so that it feeds long-edge first, if the requested page size will fit that way. The *CustomPageSize invocation code cannot be expected to override such behavior.*

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

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

This provides the allowable types and ranges for each of the custom page size parameters (*paramOption*) required by the invocation code of the *CustomPage-Size statement. There must be one *ParamCustomPageSize statement 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.

For example:

```
*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
```

The value is a StringValue with multiple components separated by white space. The value of *order* indicates the order in which the parameter named by *paramOption* 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 means a real number of PostScript default units. The print manager or application is responsible for converting user-supplied values into the correct type. For example, a value of points tells an application that, although the units might be obtained from the user in any form offered by the application, 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 *min* and *max*, representing the minimum and maximum acceptable numbers, inclusive, with the minimum value first. The type of *min* and *max* must match the *type* value for that parameter. For example, if *type* is int, then *min* and *max* must both be integers. A print manager should use the minimum and maximum values for each parameter to ensure that the user provides parameters in the valid range.

If the device does not support offsetting the image on the media, the *min* and *max* range values for WidthOffset or HeightOffset (or both, if offsetting is not supported in either direction) will both be 0 (zero). The print manager can use this information to limit or disable user selection of the offsetting feature.

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

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

On devices that support custom page sizes, these QuotedValue statements indicate the maximum media width and height allowed by the device when a custom page size is requested. Both *MaxMediaWidth and *MaxMediaHeight are expressed in PostScript default units. *MaxMediaWidth is measured perpendicular to the media feed direction and *MaxMediaHeight is measured parallel to the media feed direction.

A print manager must ensure that the sum of Width plus WidthOffset does not exceed the value of *MaxMediaWidth. Likewise, it must ensure that the sum of Height plus HeightOffset does not exceed the value of *MaxMediaHeight.

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

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

The absolute maximum width and height of media supported by the device can be obtained from the values of *MaxMediaWidth and *MaxMediaHeight respectively. However, some devices support different sizes of media cassettes, so the *current* maximum width or height might be less than the absolute maximum width or height respectively. *?CurrentMediaWidth returns a real number specifying the maximum width, in PostScript default units, of the currently installed media. *?CurrentMediaHeight returns a real number specifying the maximum height, in PostScript default units, of the currently installed media.

If these queries are available, a print manager can use them to replace the values of *MaxMediaWidth and *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 *MaxMediaWidth and *MaxMediaHeight.

***CenterRegistered:** True | False

This keyword 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 installs 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.

***LeadingEdge** *edgeOption*: " "

***DefaultLeadingEdge:** *edgeOption*

*LeadingEdge allows the user to tell the print manager how the *current* input slot has been configured to feed the page. This is both an assertion of how an input slot is set up (for cut-sheet media) and a partial request for page image orientation (for roll-fed media). See *Responsibilities of a Print Manager Regarding Custom Page Sizes* at the end of this section for a description of how the print manager can use this information to determine the orientation and imageable area of the page. *LeadingEdge should be displayed as a PickOne menu and should follow the rules for PickOne keywords, although it is not surrounded by *OpenUI/*CloseUI (see the description of *OpenUI/*CloseUI for information on PickOne). The value of *DefaultLeadingEdge provides a default state for the print manager to display.

The options for *edgeOption* are:

- Short—The currently selected input slot expects the page to be fed short-edge first, or the user would like the page image printed short-edge first on roll-fed media.

- Long—The currently selected input slot expects the page to be fed long-edge first, or the user would like the page image printed long-edge first on roll-fed media. On roll-fed media, this is often also called *transverse* or *media saving*.

- PreferLong—The currently selected input slot has been configured to rotate the page image to correspond to long-edge feed **if** the page will fit that way. That is, if Width is less than Height (which would normally produce a short-edge feed), and if Height is less than or equal to *MaxMediaWidth, the device will rotate the page image to be long-edge feed. If Width is less than Height and Height is greater than the value of *MaxMediaWidth, the page will remain short-edge feed.

- Forced—The device performs no page image rotation. The user can request a custom page size whose Width and Height define it as short-edge feed, and if the device is configured for long-edge feed, the short-edge feed image will be printed on the long-edge feed page, so clipping will probably occur. Likewise, a long-edge feed image can be printed on a short-edge feed page, with clipping equally likely to occur.

- Unknown—Nothing is known about the leading edge, so Orientation and the imageable area cannot be calculated accurately.

Only the options that are supported by the device will be listed. If a particular input slot places restrictions on the choice of leading edge, there will be *NonUIConstraints between *LeadingEdge and *InputSlot or *ManualFeed. For example, this device supports only Short and Long for *LeadingEdge, and supports only short-edge feed from the manual feed slot:

```
*LeadingEdge Short: ""
*LeadingEdge Long: ""
*DefaultLeadingEdge: Short
*NonUIConstraints: *ManualFeed True *LeadingEdge Long
*NonUIConstraints: *LeadingEdge Long *ManualFeed True
```

Note that *LeadingEdge does not control the device in any direct way; it is an assertion from the user to the print manager about how the device has been configured, and it aids the print manager in determining the value of the Orientation parameter. This keyword is in the form of an InvocationValue for convenience, but the InvocationValue quotes will be empty. This keyword will not be surrounded by *OpenUI/*CloseUI because it requires extra action on the part of a print manager and is therefore not suitable for blind parsing. A *NonUIOrderDependency statement is not necessary, as no code will be downloaded from this keyword.

Note *To print manager authors: Some print managers provide leading-edge control in the form of a two-state checkbox labeled Transverse. In that case, an empty box (Transverse Off) should cause *LeadingEdge to be set to Short, and a checked box (Transverse On) should cause *LeadingEdge to be set to Long.*

Note *To builders of PPD files: See section 6.3 for examples of how to determine which *LeadingEdge options are supported on a device and how to write *NonUIConstraints entries for *LeadingEdge. Be careful, when writing the *NonUIConstraints entries, to not exclude all options at once. At any given time, at least one option for *LeadingEdge must be available to the user.*

Cut-Sheet Keywords

The following keywords apply only to devices that can accept cut-sheet pages or can treat roll-fed pages as if they were individual sheets, imaging within an area smaller than the requested page size.

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

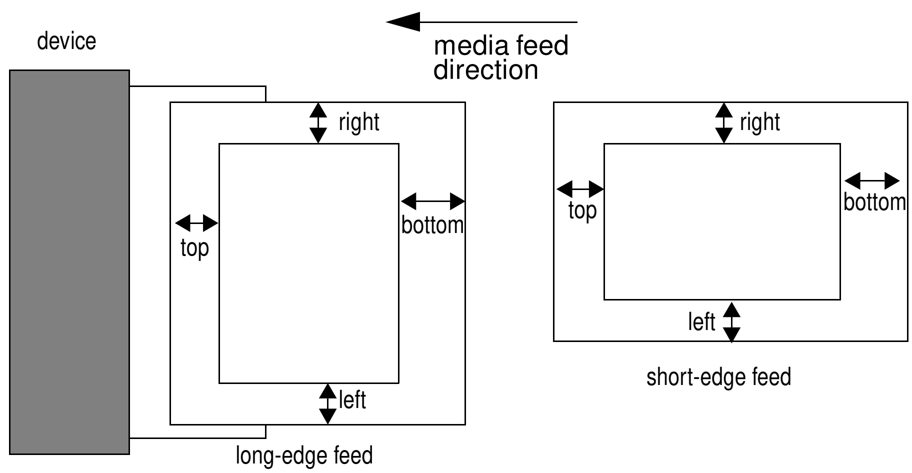
This keyword describes how much space around the outer edge of the page **cannot** be imaged because of hardware restrictions. A print manager can use this information to calculate the imageable area and tell a user when the entire requested custom page size cannot be printed upon, or to show the user which part of the page can be imaged. For non-custom page sizes, the keyword *ImageableArea provides the same information (in the form of imageable area, rather than non-imageable area) for each supported page size. See *Responsibilities of a Print Manager Regarding Custom Page Sizes* at the end of this section for a description of how the print manager can use *HWMargins.

The StringValue of this keyword is composed of multiple components, separated by white space. The components are real numbers of 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 (the leading edge in the direction of media feed).

right, *left,* and *bottom* are similarly defined, as shown in Figure 4. This is a top view; you are looking down at the paper going into the device.

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 statement to describe this would be

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

Any or all of the values may be 0 (zero). 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.

This keyword will be present only if the device supports custom page sizes and has hardware-imposed margins or can be configured to behave as though it has hardware-imposed margins, imaging in an area that is smaller than the requested page size.

Note *To builders of PPD files: Typically, PPD files for devices that accept only cut-sheet media will have* *HWMargins*. Typically, PPD files for devices that accept only roll-fed media will not have* *HWMargins*. Devices that accept only roll-fed media are usually able to image over the entire requested page area; any restrictions on page size due to hardware control mechanisms are described by* *MaxMediaWidth *and* *MaxMediaHeight*. However, if the device supports both roll-fed media and cut-sheet media, or if the device supports only roll-fed media but can be configured to image within a smaller area as if the pages were individual sheets,* *HWMargins *will be needed in the PPD file. See the description of* *UseHWMargins *in this section.*

Note *Because of varying margin widths, the interaction of custom page sizes with duplexing (or other operations that may shift the image on the page) on cut-sheet media is unpredictable.*

***UseHWMargins** True | False: " "

***DefaultUseHWMargins:** True | False

The presence of *UseHWMargins indicates a device that can switch between imaging over the entire page area (typical roll-fed media behavior) and imaging only in the area dictated by *HWMargins (typical cut-sheet media behavior). *UseHWMargins allows the user to tell the print manager how to define the imageable area of the requested custom page size. Although not surrounded by *OpenUI/*CloseUI, *UseHWMargins should be displayed and treated as a Boolean feature. (See *OpenUI/*CloseUI for a description of Boolean.)*DefaultUseHWMargins provides a default state for *UseHWMargins for the print manager to display. True means the print manager should treat the requested page as a cut-sheet page with hardware-imposed margins, and should use *HWMargins to calculate the imageable area of the page. False means the print manager should treat the requested page as a roll-fed page, imaging over the entire area of the page.

Note that *UseHWMargins does not control the device in any way; it is merely a request from the user to the print manager for a specific action. The print manager should warn the user that the device must be set up properly to achieve the correct result, as many devices require user interaction at the device's front panel to establish cut-sheet behavior vs. roll-fed behavior.

This keyword is in the form of an InvocationValue for convenience, but the InvocationValue quotes will be empty. A *NonUIOrderDependency statement is not necessary, as no code will be downloaded from this keyword. This keyword will not be surrounded by *OpenUI/*CloseUI because it requires extra action on the part of a print manager and is therefore not suitable for blind parsing. If cut-sheet behavior is only available through certain input slots, there will be *NonUIConstraints between *UseHWMargins and *InputSlot or *ManualFeed. If the device cannot provide both types of imaging (entire page area vs. smaller imageable area), this keyword will be omitted. If *UseHWMargins is present, *HWMargins must also be present.

Note *To print manager authors: If *HWMargins is missing from the PPD file, assume that all four values are zero. If *HWMargins is present and *UseHWMargins is missing, assume that *HWMargins should always be used.*

Note *To builders of PPD files: Usually, *UseHWMargins will not be present in PPD files for devices that accept only cut-sheet media, because such devices can image only within the smaller area imposed by *HWMargins, so there is no choice for the user to make about using *HWMargins. Likewise, *UseHWMargins (and *HWMargins) will not usually be present in PPD files for devices that accept only roll-fed media, because such devices usually cannot impose a smaller imageable area on the requested page size.*UseHWMargins will usually be needed only in the PPD file of a device that supports both roll-fed and cut-sheet media, or a device that supports only roll-fed media but can be configured to image within a smaller area as if the pages were individual sheets, or a device that supports only cut-sheet media but can be configured to image across the entire page area. In these cases, there is a choice of imaging methods and the user must tell the print manager about the device's current configuration or how the page should be imaged.*

Responsibilities of a Print Manager Regarding Custom Page Sizes

If *UseHWMargins is present, it should be offered as a two-state menu or check-box in the custom page size user interface. *LeadingEdge, if present, should be offered as a PickOne type of menu. Throughout the user selection process, the print manager must consult any *NonUIConstraints statements for *UseHWMargins, *CustomPageSize, and *LeadingEdge.

If *UseHWMargins is True, the print manager should warn the user that the device may require setup at the device's front panel. If *UseHWMargins is not present and *LeadingEdge is changed by the user from its default state, the print manager should warn the user that the device may require adjustment of the input trays.

When obtaining the values for Width, Height, WidthOffset, HeightOffset, and Orientation from the user, the print manager must

- convert the value to the appropriate units listed under *ParamCustomPageSize for that parameter, if necessary. For example, Width and Height may be obtained from the user in inches or millimeters and must be converted to points before any further calculations occur.

- ensure that each value falls within the appropriate range listed under *ParamCustomPageSize for that parameter. If the range is limited to a single choice (for example, the range for Orientation may be 0..0), the print manager might wish to prevent the user from typing values in that field in the user interface.

- if *CenterRegistered is True, the print manager should warn the user to provide the correct value for WidthOffset if the installed media is narrower than the transport mechanism.

- ensure that the sum of Width plus WidthOffset does not exceed the value of *MaxMediaWidth. Likewise, it must ensure that the sum of Height plus HeightOffset does not exceed the value of *MaxMediaHeight.

Emitting the correct *CustomPageSize parameters in the correct order can be complicated. Here are the key points that must be considered:

- If *LeadingEdge is Short, Long, or PreferLong, the device will rotate the page image in device space so that the long axis is parallel to the long axis of the physical page. To calculate the imageable area correctly, the print manager must generate Width and Height so that they match the physical page orientation. If the user sets *LeadingEdge to Long, Width must be greater than or equal to Height. Likewise, if the user sets *LeadingEdge to Short, Height must be greater than or equal to Width. Depending on how the print manager presents the dimensions of the page to the user, the print manager may have to perform some manipulation on the dimensions to produce the correct Width and Height before placing them on the operand stack.

- If *LeadingEdge is PreferLong: If Width ≥Height, or if Width < Height and Height ≤ *MaxMediaWidth, the print manager should behave as if *LeadingEdge is Long when consulting Table 2. If Width < Height and Height >*MaxMediaWidth, the print manager should behave as if *LeadingEdge is Short when consulting Table 2.

- If *LeadingEdge is Forced, the print manager must accept the values of Width and Height as provided by the user, and set *LeadingEdge accordingly for its internal use. If Width < Height, *LeadingEdge should be Short. If Width ≥ Height, *LeadingEdge should be Long. The print manager may also wish to warn the user that if the device is not set up accordingly, the choice of Forced may result in the clipping and apparent rotation of the image.

- If *LeadingEdge is Unknown, see the note after Table 3.

- Unless the print manager lets the user enter Orientation directly, the print manager must deduce the value of Orientation from a combination of user requests, as shown in Table 2.

Table 2 *Determining the value of* Orientation

User chooses:	Dimensions	Portrait (x < y)	Landscape (y < x)
*LeadingEdge: Long	Width > Height	Orientation = 0 or 2	Orientation = 1 or 3
*LeadingEdge: Short	Height > Width	Orientation = 1 or 3	Orientation = 0 or 2

- Most print managers restrict the choices for Orientation to 0 and 1, and must choose accordingly from Table 2. To offer the user a choice of 2 or 3 for Orientation, the print manager must offer a user interface that allows the user to choose all four orientations pictorially or by entering the value directly.

Once the value of Orientation has been determined, the print manager can decide whether or not it is necessary to calculate and display the imageable area, using the following algorithms:

- If *HWMargins is not present or if *UseHWMargins is False, the page will be imaged over its entire imageable area, so no imageable area calculations are necessary. The print manager may skip to the last step in this section.

- If *HWMargins is present and *UseHWMargins is not present or is True, and all *NonUIConstraints conflicts have been resolved, then the print manager can calculate the imageable area of the custom page size and show a pictorial to the user. Using Orientation as a key, the print manager can use Table 3 to calculate the imageable area. The imageable area is expressed as the *x* and y coordinates of the lower left and upper right corners of the imageable area. These coordinates are referred to as ll_x, ll_y, ur_x, and ur_y, respectively. *top, bottom, left,* and *right* are defined in the description of *HWMargins.

Table 3 *Using Orientation and *HWMargins to determine imageable area*

Orientation	ll_x	ll_y	ur_x	ur_y
0	*top*	*left*	Height minus *bottom*	Width minus *right*
1	*left*	*bottom*	Width minus *right*	Height minus *top*
2	*bottom*	*right*	Height minus *top*	Width minus *left*
3	*right*	*top*	Width minus *left*	Height minus *bottom*

Note *Previous versions of this specification, which did not use Orientation to determine the correspondence of image edge to page edge, recommended using the simpler but less accurate method of subtracting the largest of the four *HWMargins values from each edge of the page. This method may produce a much smaller imageable area than the device is capable of handling, but it does guarantee that all marks made in the calculated imageable area will be visible. For maximum user satisfaction, print managers should be written to use the newer, more accurate method, using Orientation as a key. However, if *LeadingEdge is Unknown, the print manager cannot calculate Orientation and must resort to a fallback position such as described above.*

Finally, the print manager must ensure that the parameters are placed on the operand stack in the proper order (documented by *ParamCustomPageSize) and followed by the invocation code from *CustomPageSize.

5.17 Media Handling Features

The keywords in this section provide handling of media other than media selection, such as output attributes.

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

***DefaultOutputBin:** *binOption* | Unknown

***?OutputBin:** *"query"* (returns: *binOption* | Unknown)

*OutputBin provides the InvocationValue to select different output paths for media. *DefaultOutputBin denotes the default output path. *?OutputBin returns a string denoting the current output path. If the device does not provide software-selectable output paths, these keywords will be omitted.

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.

Note *To builders of PPD files: Although older PPD files (and the tools that built them) often included *DefaultOutputBin, it provides no useful information to a print manager unless the complete *OutputBin entry is also present. If the output bins are not software-selectable, omit these keywords.*

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

***DefaultOutputOrder:** *orderOption* | Unknown

***?OutputOrder:** *"query"* (returns: *orderOption* | Unknown)

*OutputOrder 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 indicates the default page stacking order of the default output bin. *?OutputOrder returns a string denoting the current page stacking order of the current output bin.

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.

Note *To builders of PPD files: *DefaultOutputOrder can be used by a print manager to determine in which order to send the pages of the job, so it should usually be included in the PPD file even when the output order cannot be changed. If *DefaultOutputOrder is stand-alone, its value must be Normal or Reverse. See the note under Unknown in section 4.4, and section 4.5.*

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

This is an informational statement that indicates the page stacking order of each output bin. It is useful only if the device has multiple software-selectable output bins. The option keyword *binOption* must be a valid option keyword listed under *OutputBin. The StringValues 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 keyword 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

***DefaultTraySwitch:** True | False | Unknown

***?TraySwitch:** *"query"* (returns: True | False | Unknown)

*TraySwitch 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 denotes the default state of the automatic tray switching mechanism. *?TraySwitch returns the current state of tray switching.

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

***DefaultSignature:** *signatureOption* | Unknown

***?Signature:** "*query*" (returns: *signatureOption* | Unknown)

*Signature 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. *DefaultSignature denotes the default state of the automatic signature feature. *?Signature returns a string denoting the current state of the automatic signature feature.

One of the *signatureOption*s must be None or False, to turn off the automatic signature feature. Other 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.

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

***DefaultDuplex:** *duplexOption* | Unknown

***?Duplex:** *"query"* (returns: *duplexOption* | Unknown)

*Duplex provides the InvocationValue to control the duplex (two-sided printing) feature. *DefaultDuplex denotes the default state of the duplex feature. *?Duplex returns a string denoting the current state of the duplexing mechanism.

The currently registered values for *duplexOption* are listed below. One of the options must be None or False, for "no duplexing" (that is, produce simplex or one-sided printing). *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

***DefaultOutputMode:** *modeOption* | Unknown

***?OutputMode:** *"query"* (returns: *modeOption* | Unknown)

*OutputMode provides the InvocationValues 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 denotes the default output mode. *?OutputMode returns a string denoting the current output mode.

5.18 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 file should be grouped by *OpenGroup/*CloseGroup. For a complete example, refer to the sample PPD files in section 6.

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

***DefaultCollate:** *collateOption* | Unknown

***?Collate:** *"query"* (returns: *collateOption* | Unknown)

*Collate 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 denotes the default state of the collator mechanism. *?Collate returns a string denoting the current state of the collator mechanism.

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

***DefaultFoldType:** *foldOption* | Unknown

***?FoldType:** "*query*" (returns: *foldOption* | Unknown)

*FoldType provides the InvocationValue to control which type of fold is invoked, if any. *DefaultFoldType denotes the default type of fold. *?FoldType returns a string denoting the current type of fold.

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	None

One of the options must be None or False, to turn off folding. Builders of PPD files should include the following *UIConstraints statements to disable *FoldType unless *FoldWhen has been invoked with a value other than None:

```
*UIConstraints: *FoldWhen None *FoldType
*UIConstraints: *FoldType None *FoldWhen
```

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

***DefaultFoldWhen:** *foldOption* | Unknown

***?FoldWhen:** "*query*" (returns: *foldOption* | Unknown)

*FoldWhen provides the InvocationValue to control when a job is folded, if folding has been invoked. *DefaultFoldWhen denotes the default state of when the job will be folded. *?FoldWhen returns a string denoting the current state of *FoldWhen. The following *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 **NumCopies** and **Collate** in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfPage—Fold after each **showpage** or **copypage**.

One of the options must be None or False, to turn off folding. Builders of PPD files should include the following *UIConstraints statements to disable *FoldType unless *FoldWhen has been invoked with a value other than None:

```
*UIConstraints: *FoldWhen None *FoldType
*UIConstraints: *FoldType None *FoldWhen
```

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

***DefaultSorter:** *sortOption* | Unknown

***?Sorter:** "*query*" (returns: *sortOption* | Unknown)

*Sorter 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.

The currently registered values for *sortOption* are:

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

One of the options must be None or False to turn off sorting. *DefaultSorter denotes the default state of the sorter mechanism. *?Sorter returns a string denoting the current state of the sorter mechanism.

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

***DefaultStapleLocation:** *stapleOption* | Unknown

***?StapleLocation:** "*query*" (returns: *stapleOption* | Unknown)

*StapleLocation 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. *DefaultStapleLocation denotes the default location for stapling. *?StapleLocation returns a string that denotes the current stapling location.

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.
- None—No stapling.

One of the options must be None or False, to turn off stapling. Builders of PPD files should include the following *UIConstraints statements to disable *StapleLocation unless *StapleWhen has been invoked with a value other than None:

```
*UIConstraints: *StapleWhen None *StapleLocation
*UIConstraints: *StapleLocation None *StapleWhen
```

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

***DefaultStapleX:** *stapleOption* | Unknown

***?StapleX:** "*query*" (returns: *stapleOption* | Unknown)

*StapleX 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.

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.
- None—No stapling.

One of the options must be None or False, to turn off stapling. Builders of PPD files should include the following *UIConstraints statements to disable *StapleX unless *StapleWhen has been invoked with a value other than None:

```
*UIConstraints: *StapleWhen None *StapleX
*UIConstraints: *StapleX None *StapleWhen
```

*DefaultStapleX denotes the default location for stapling. *?StapleX returns a string that denotes the current stapling location.

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

***DefaultStapleY:** *stapleOption* | Unknown

***?StapleY:** "*query*" (returns: *stapleOption* | Unknown)

*StapleY 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.

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
- None—No stapling.

One of the options must be None or False, to turn off stapling. Builders of PPD files should include the following *UIConstraints statements to disable *StapleY unless *StapleWhen has been invoked with a value other than None:

```
*UIConstraints: *StapleWhen None *StapleY
*UIConstraints: *StapleY None *StapleWhen
```

*DefaultStapleY denotes the default location for stapling. *?StapleY returns a string that denotes the current stapling location.

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

UI

***DefaultStapleWhen:** *stapleOption* | Unknown

***?StapleWhen:** "*query*" (returns: *stapleOption* | Unknown)

*StapleWhen provides the InvocationValue to control when a job is stapled. Examples include "end of job," "end of group." One of the options must be None or False to turn off stapling. See the descriptions of *StapleX, *StapleY, *StapleOrientation, and *StapleLocation for examples of *UIConstraints that should be written for *StapleWhen.

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 **NumCopies** and **Collate** in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfPage—Staple after each **showpage** or **copypage**.

*DefaultStapleWhen denotes the default time for stapling. *?StapleWhen returns a string that denotes when stapling will occur under the current setting.

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

UI

***DefaultStapleOrientation:** *orientationOption* | Unknown

***?StapleOrientation:** "*query*" (returns: *orientationOption* | Unknown)

*StapleOrientation provides the InvocationValue to control the orientation of the staple; for example, 45 degrees.These *orientationOptions* 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.

- None—No specific staple orientation requested (empty code).

One of the options must be None or False. Builders of PPD files should include the following *UIConstraints statements to disable *StapleOrientation unless *StapleWhen has been invoked with a value other than None:

```
*UIConstraints: *StapleWhen None *StapleOrientation
*UIConstraints: *StapleOrientation *StapleWhen None
```

*DefaultStapleOrientation denotes the default orientation for the staple. *?StapleOrientation returns a string that denotes the current staple orientation.

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

***DefaultBindEdge:** *bindOption* | Unknown

***?BindEdge:** "*query*" (returns: *bindOption* | Unknown)

*BindEdge provides the InvocationValue to control which edge is bound. *DefaultBindEdge denotes the default edge for binding. *?BindEdge returns a string denoting which edge will be bound under the current setting.

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.

- None—No binding.

One of the options must be None or False, to turn off binding. Builders of PPD files should include the following *UIConstraints statements to disable *BindEdge unless *BindWhen has been invoked with a value other than None:

```
*UIConstraints: *BindWhen None *BindEdge
*UIConstraints: *BindEdge None *BindWhen
```

***BindType** *bindtypeOption*: *"invocation"* UI

***DefaultBindType:** *bindtypeOption* | Unknown

***?BindType:** "*query*" (returns: *bindtypeOption* | Unknown)

*BindType provides the InvocationValue to control the type of binding. *DefaultBindType denotes the default type of binding.*?BindType returns a string indicating which type of binding will occur given the current setting. *bindtypeOption* is a product-dependent string describing the type of binding available (for example, Spiral). One of the options must be None or False, to disable binding. Builders of PPD files should write *UIConstraints entries to disable *BindType when *BindWhen is None; see *BindEdge for an example.

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

***DefaultBindColor:** *colorOption* | Unknown

***?BindColor:** "*query*" (returns: *colorOption* | Unknown)

*BindColor provides the InvocationValues to control the binding color. *DefaultBindColor denotes the default color of binding. *?BindColor returns a string indicating which binding color will be used under the current setting. The valid values for *colorOption* are product-dependent strings describing the color of the binding, such as Blue or Red. One of the options must be None or False. Builders of PPD files should write *UIConstraints entries to disable *BindColor when *BindWhen is None; see *BindEdge for an example.

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

***DefaultBindWhen:** *bindOption* | Unknown

***?BindWhen:** *"query"* (returns: *bindOption* | Unknown)

*BindWhen provides the InvocationValue to turn on binding and to control when a job is bound. *DefaultBindWhen denotes the default time for binding. *?BindWhen returns a string that denotes when binding will occur under the current setting.

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 **NumCopies** and **Collate** in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.

One of the options must be None or False to turn off binding. See the description of *BindEdge for an example of *UIConstraints that should be written between *BindWhen and *BindColor, *BindType, and *BindEdge.

***Booklet** *bookletOption*: *"invocation"* UI

***DefaultBooklet:** *bookletOption* | Unknown

***?Booklet:** "*query*" (returns: *bookletOption* | Unknown)

*Booklet 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.

*DefaultBooklet denotes the default state of booklet making. *?Booklet returns a string denoting the current state of booklet making.

***Slipsheet** *slipsheetOption*: "*invocation*"

***DefaultSlipsheet:** *slipsheetOption* | Unknown

***?Slipsheet:** "*query*" (returns: *slipsheetOption* | Unknown)

UI

*Slipsheet 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. A *job* is defined in section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfSet—Insert slipsheet at the end of the current set. For a definition of "set," see **NumCopies** and **Collate** in Table 4.11 of the *PostScript Language Reference Manual, Second Edition*.
- EndOfPage—Insert slipsheet at the end of the current page.
- 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 denotes the default state of slipsheeting. *?Slipsheet returns a string denoting the current state of slipsheeting.

***InsertSheet** True | False: *"invocation"*

***DefaultInsertSheet:** True | False | Unknown

***?InsertSheet:** *"query"* (returns: True | False | Unknown)

*InsertSheet provides the InvocationValue to insert a sheet at a specific place in the document. True means that the next page will be drawn from a special input tray and inserted in the page sequence. False means that the next page will be drawn from the regular input tray. 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 denotes the default state of *InsertSheet. *?InsertSheet returns a string denoting the current state of *InsertSheet.

Note *To builders of PPD files: This feature was marked with the "UI" symbol in previous versions of this specification. Upon further study, it was found that this keyword should not be surrounded by *OpenUI/*CloseUI because the print manager must do more than blindly post the feature and execute the code; it must provide a method for the user to specify where the page should be inserted and then perform the procedures described above.*

***Jog** *jogOption: "invocation"* UI

***DefaultJog:** *jogOption* | Unknown

***?Jog:** *"query"* (returns: *jogOption* | Unknown)

*Jog 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 denotes the default state of jogging. *?Jog returns a string denoting the current state of jogging.

5.19 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, with the exception of *ReferencePunch, is documented in section 4.11 of the *PostScript Language Reference Manual, Second Edition*.

***MirrorPrint** True | False: *"invocation"*

***DefaultMirrorPrint:** True | False | Unknown

***?MirrorPrint:** *"query"* (returns: True | False | Unknown)

*MirrorPrint provides the InvocationValue to turn the mirror print feature on (True) and off (False). *DefaultMirrorPrint denotes the default state of mirror printing. *?MirrorPrint returns the current setup for mirror printing.

***NegativePrint** True | False: *"invocation"*

***DefaultNegativePrint:** True | False | Unknown

***?NegativePrint:** *"query"* (returns: True | False | Unknown)

*NegativePrint provides the InvocationValue to turn the negative print feature on (True) and off (False).*DefaultNegativePrint denotes the default state of negative printing. *?NegativePrint returns the current setup for negative printing.

***AdvanceMedia** *advanceOption*: *"invocation"*

***DefaultAdvanceMedia:** *advanceOption* | Unknown

***?AdvanceMedia:** *"query"* (returns: *advanceOption* | Unknown)

*AdvanceMedia 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 denotes the default state of *AdvanceMedia. The query *?AdvanceMedia returns a string denoting the current state of *AdvanceMedia.

***CutMedia** *cutOption*: *"invocation"*

UI

***DefaultCutMedia:** *cutOption* | Unknown

***?CutMedia:** *"query"* (returns: *cutOption* | Unknown)

*CutMedia provides the InvocationValue to tell the device when to cut roll-fed media. *DefaultCutMedia denotes the default state of *CutMedia. *?CutMedia returns a string denoting the current state of *CutMedia.

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.**

***ReferencePunch** *mediaOption*: *x y*

Printing plates are typically punched along their leading edge to facilitate mounting on printing presses. Some devices provide an equivalent punch system for film so that the film may be accurately contacted to a printing plate. This keyword provides the location of the reference punch (the center point of the center hole on the punch rack) so that an application may position the image on the film relative to the punch.

The parameter *mediaOption* designates the page size. It must be one of the option keywords listed under *PageSize and *PageRegion in the PPD file. There must be one *ReferencePunch statement for every page size that can be punched.

The parameters *x* and *y* provide the coordinates, in default user space, of the reference punch. For example, if the punch rack was 1/8 inch ahead of the page image along the media feed direction, and the reference punch was centered along the page in the direction perpendicular to media feed direction, the PPD file would contain the following statements for the page sizes Letter and Letter.Transverse:

```
*ReferencePunch Letter: 792.125 306
*ReferencePunch Letter.Transverse: -0.125 396
```

This example assumes that, on this device, Letter has a width of 612, a height of 792, and an orientation of 1, and Letter.Transverse has a width of 792, a height of 612, and an orientation of 0, where width, height, and orientation are defined as in Figure 3 on page 109.

*Note: There is currently no *ReferencePunch support for custom page sizes.*

5.20 Font Related Keywords

This section contains keywords that provide information about the fonts on the device.

***FDirSize:** *int*

This provides the size, in bytes, of the font directory in the device's interpreter. This is only useful for Level 1 devices, which have a fixed-size font directory. This keyword will not appear in the PPD file of a Level 2 device. If a print manager is keeping track of how many fonts have been downloaded to the device, this StringValue tells a print manager when the directory is getting full, so the print manager can flush out old fonts to make room for new ones. Without this information, a print manager may either flush too often or encounter a **dictfull** error.

***FCacheSize** *vmOption*: *int*

This StringValue provides the size of the font cache, in bytes, for a given level of memory installed in the device. *vmOption* must be a valid option listed under *VMOption and *InstalledMemory. (See the description of those keywords for details on *vmOption*.) If the device accepts installable memory modules, there should be a *FCacheSize statement for each module size.

***Font** *fontname*: *encoding "(version)" charset status*

This keyword provides one line of information for each font that may be resident on the product. (To understand which fonts are listed, see *Listing fonts* at the end of this keyword description.) The option *fontname* is the valid PostScript language name of the font, without the leading slash. The StringValue has multiple components separated by white space.

Value of *encoding*

The *encoding* value has slightly varying meanings depending on the font type. If the encoding cannot be determined, the value of *encoding* may be Unknown.

For Roman (one byte per character) fonts, the *encoding* value indicates the default encoding of each font. Fonts are usually re-encoded by applications or print managers to provide other encodings; the *charset* value (described later) for each font indicates which encodings are possible for that font.

The following are the currently defined *encoding* values for Roman fonts:

- Standard—This font, by default, uses the Adobe **StandardEncoding** vector.
- Special—This font has a nonstandard font-specific encoding (for example, the font named Sonata, which is composed of musical symbols).
- ISOLatin1—This font, by default, uses the Adobe **ISOLatin1Encoding** vector.
- Expert—This font, by default, uses the Adobe Expert encoding vector.
- ExpertSubset—This font, by default, uses the Adobe ExpertSubset encoding vector.

Older composite fonts use the following *encoding* values:

- JIS—A Japanese font with JIS (Japan Industrial Standard) encoding. (This is a two byte-per-character encoding.)
- RKSJ—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—A Japanese font with EUC (Extended UNIX Code) encoding. (This is a two byte-per-character encoding.)
- Shift-JIS—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.)

CID-keyed composite fonts use the *encoding* value to record the CMap component of the font. For details on CMaps in CID-keyed composite fonts, including the CMap names for Chinese and Korean fonts, see Technical Note #5094, *Adobe CJK Character Collections and CMaps for CID-Keyed Fonts*, available from the Adobe Developers Association. The following is a list of the currently registered values for *encoding* for Japanese CID-keyed fonts:

78-H	78-V	78-RKSJ-H	78-RKSJ-V
78-EUC-H	78-EUC-V	83pv-RKSJ-H	
90pv-RKSJ-H	90ms-RKSJ-H	90ms-RKSJ-V	
Add-RKSJ-H	Add-RKSJ-V	Add-H	Add-V
Ext-RKSJ-H	Ext-RKSJ-V	Ext-H	Ext-V
EUC-H	EUC-V	RKSJ-H	RKSJ-V
Hankaku	Hiragana	H	V
Katakana	Roman	WP-Symbol	
NWP-H	NWP-V		

Note *The encodings named NWP-H and NWP-V are obsolete and may be removed from a future version of this specification.*

Note *To builders of PPD files: The currently registered values for encoding are primarily for the most common Japanese fonts. If a font has a CMap name that is not listed here, the appropriate CMap name from Technical Note #5094 should be inserted in the encoding field. However, print managers, depending on how they use the encoding field, might not recognize new encoding values until they are listed in this specification and the print manager is updated.*

Value of *version*

The value of *version* is the version number of the font; for most fonts, it is the value of the key **Version** in the **FontInfo** dictionary that is a subdictionary of the font dictionary. For CID-keyed composite fonts, it is the value of the key **CIDFontVersion** in the dictionary for the **CIDFont** resource instance.

Value of *charset*

The *charset* value of the *Font keyword indicates which shape descriptions (glyphs) are contained in the font and are available for re-encoding. If this information cannot be determined, the value of *charset* may be Unknown.

Valid *charset* values for Western (Roman) fonts are:

- Standard—This indicates a Roman font that contains the character set that supports both the Standard and ISOLatin1 encodings. Most Roman fonts from Adobe will have this value in the *charset* field of their *Font statements.

- 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.

Older composite fonts use the following *charset* values:

- JIS-83—Supports the JIS X0208-1983 character set.

- JIS-78—Supports the JIS 1978 character set.

- 83pv—Supports the 83pv (Apple® Macintosh-compatible) character set.

- Add—Supports the Add (Fujitsu FM system-compatible) character set.

- Ext—Supports the Extended (NEC PC-98-compatible) character set.

- NWP—Supports the NWP (NEC Word Processor) character set.

CID-keyed composite fonts use the *charset* value to record the *registry*, *ordering*, and *supplement* values of the font. For details on CID-keyed fonts, see Technical Note #5094, *Adobe CJK Character Collections and CMaps for CID-Keyed Fonts*, available from the Adobe Developers Association. When creating a *charset* value, the registry, ordering, and supplement fields are separated by hyphens. The following is a list of the currently registered values for *charset* for CID-keyed fonts:

Adobe-Japan1-0	Adobe-Japan1-1	Adobe-Japan1-2
Adobe-Japan2-0	Adobe-Korea1-0	Adobe-Korea1-1
Adobe-GB1-0	Adobe-CNS1-0	

Note *To builders of PPD files: If a font contains a registry, ordering, and supplement combination that does not appear on this list, you may create a charset value composed of the registry, ordering, and supplement values from the font, in that order, separated by hyphens. Some print managers might not recognize such newly-created charset values; depending on whether or not the print manager uses the charset information, this might not matter.*

Value of *status*

The *status* field indicates whether or not the font can be removed without causing the printer to cease its normal functioning.Valid values 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.

Table 4 contains examples of font distribution methods and associated *status* keywords. This table is not exhaustive as to the different methods of font distribution.

Table 4 *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

While most devices include fonts in ROM, a device could ship with all fonts having a *status* of Disk. For example, all of a device's fonts could be shipped on an external CD-ROM.

Listing fonts

All valid font dictionaries found on the device will have *Font statements; the list is not limited to Type 1 fonts. All fonts shipped with the product in its minimal configuration are listed. These fonts may be in ROM or on a peripheral device such as a hard disk, as long as they are always shipped with the product.

Note *To builders of PPD files: If additional fonts are available on a plug-in cartridge, hard disk, or similar peripheral device that does* ***not*** *ship with the product in its minimal configuration, a separate PPD file should be created to represent the primary device with the peripheral device attached. For example, there might be a PPD file for "Acme FunPrinter" and a separate PPD file for "Acme FunPrinter with Display Font Cartridge". The second PPD file would be a duplicate of the first PPD file, except that the second PPD file would contain extra *Font statements to list the fonts available on the*

Display Font Cartridge. Theoretically, this could also be accomplished by creating a local customization file with the extra fonts listed, but support for the installation of local customization files is extremely limited or nonexistent in most common operating environments, so this has little practical value. See the next note for long-term plans to make the inclusion of aftermarket font devices more streamlined.

Note *To print manager authors: Although it is not legal now, in a future edition of this specification, it will be legal to use *NonUIConstraints to constrain *Font. For example, if *Option1 represents a plug-in font cartridge in the InstallableOptions group, a future version of this specification will allow*

```
*NonUIConstraints: *Option1 False *Font Palatino
```

*which would tell a print manager that the Palatino font is not available if the *Option1 font cartridge is not installed on the device. The PPD file could contain *Font entries for fonts available on peripheral devices for this product, along with the appropriate *NonUIConstraints statement that ties the presence of the font to the presence of the peripheral device. If you are writing or upgrading an application that reads *Font statements, we recommend that you include support for this future feature. That is, instead of assuming that all fonts listed under *Font are always present on the device, applications should be written to check for *NonUIConstraints on *Font and only register the font as available if the appropriate peripheral device has been registered as installed.*

Fonts that are later downloaded to the device from the host via software are usually monitored by system software (the print manager, a font downloading utility, or any application) and are not covered by this specification.

***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 statement 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 42).

The following is a look at two valid returns from the query:

```
/Optima/Optima-Bold/Optima-Oblique/Optima-BoldOblique/Courier/Symbol
*
```

and

```
/Courier
/Symbol
/Times-Roman
*
```

Note *To builders of PPD files: This keyword can return a large amount of data. If the host or communication channel cannot retrieve the data fast enough, the device's output buffer may overflow, causing data to be lost before it can be retrieved. If the device ships with a large number of fonts or will regularly be attached to a mass storage device containing many fonts, the *?FontList code should be tested over all available communication channels. If data is lost, the*?FontList code should be altered to slow down the output. One method is to output the font names in groups separated by small time delays.*

***?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 print 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.

For example:

```
/Times-Roman:Yes
/Optima:Yes
/CircleFont:No
/Adobe-Garamond:No
*
```

Note *To print manager authors: The query provided by* *?FontQuery *is often preferable to the* *?FontList *query, since that query can return a very long list of fonts in some devices, such as those with access to built-in hard disks or network font servers.*

Note *To builders of PPD files: Given a large list of font names to query, this keyword could return a large amount of data, although not typically as large as* *?FontList*. If the host or communication channel cannot retrieve the data fast enough, the device's output buffer may overflow, causing data to be lost before it can be retrieved. If the device ships with a large number of fonts or will regularly be attached to a mass storage device containing many fonts, the* *?FontQuery *code should be tested with a large list of font names over all available communication channels. If data is lost, the* *?FontQuery *code should be altered to slow down the output. One method is to output the responses in smaller groups separated by small time delays.*

5.21 Printer Messages

In an environment where 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:** *"text"*

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 statements 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 statement 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 3.5 for details on translation string syntax.

*Status: *“text”*

This lists the possible responses to a status query as QuotedValues. A status query is typically accomplished by sending ^T (control-T, decimal 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: cvr opn; source: serial25]%%
```

The statements 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). This effectively provides a list of the names of the communications channels on the device, plus any other possible sources for jobs. The following are example statements 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:** *"text"*

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 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 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 statements are optional. If a statement does not exist for a specific color, the default value should be used. For example, there might be statements for screen frequencies and screen angles for a color but not a statement 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 keywords 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 statement 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 keyword provides the transfer function InvocationValue appropriate for the given color separation keyword.

***CustomCMYK** *inkname: "cyan magenta yellow black"*

This keyword 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 statements of this sort in a PPD file. For some devices, in fact, these statements can be put into a separate file that references the original PPD file with the *Include convention, discussed 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

***DefaultSeparations:** True | False | Unknown

***?Separations:** *"query"* (returns: True | False | Unknown)

If the device can provide automatic generation of color separations, *Separations provides the InvocationValue to tell the device to output either color separations (True) or composite color (False). True means that the 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. *DefaultSeparations denotes the default state of the automatic separations mechanism.*?Separations will return the current state of the separations mechanism

Color separations are explained in section 4.8 of the *PostScript Language Reference Manual, Second Edition*. The print manager may wish to provide a more complex user interface for the user to declare which pages should be produced as separations and which should not, in which case this keyword should not be treated as a simple UI keyword.

5.23 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 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 by *SymbolLength 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 statement 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 older parsers. It is intended for use only when the code is very large—on the order of tens of kilobytes. Symbol names should not be used as the values of keywords that existed in version 3.0 of this specification or for commonly-referenced keywords, such as *PageSize.*

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

This keyword tells a parser 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 *symbolName* used in the associated *SymbolValue statement.

The StringValue components *bytecount* and *EOLcount* are unsigned integers, separated by white space. 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 statement 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 statement 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 statement either later in the PPD file or in an included PPD file.

The following two examples are both valid.

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

*%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 statement 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 Saturated.Bond.Dot: ^ColorRenderDictSaturated.Bond.Dot
...
*SymbolValue ^ColorRenderDictSaturated.Bond.Dot: "..."
```

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 statement.

6 Sample PPD File Structure

This section contains examples of

- a generic Level 2 color printer PPD file
- a generic Level 2 imagesetter PPD file
- custom page size entries for various devices

6.1 Level 2 Color Printer

This PPD file describes a Level 2 color printer with one resolution, one regular input slot, one manual feed slot, two output bins, an optional envelope feeder, and three supported page sizes. The printer can have a hard disk attached to it. It supports duplexing, choosing the media by type, and HP LaserJet emulation. It can also do color separations at the printer, when sent a composite color file.

```
*PPD-Adobe: "4.3"
*FormatVersion: "4.3"
*FileVersion: "1.0"
*LanguageEncoding: ISOLatin1
*LanguageVersion: English
*Product: "(Acme Color Printer)"
*PSVersion: "(2017.0) 0"
*Manufacturer: “Acme”
*ModelName: "Acme Color Printer v.2017"
*ShortNickName: “Acme Color Printer”
*NickName: "Acme Color Printer v.2017"
*PCFileName: “ACCOLOR1.PPD”

*%=== Basic Capabilities ===============
*LanguageLevel: “2”
*ColorDevice: True
*DefaultColorSpace: CMYK
*FreeVM: "8134935"
*FileSystem: True
*?FileSystem: "
   save false
       (%disk?%)
       { currentdevparams dup /Writeable known
         { /Writeable get {pop true} if }  { pop } ifelse
       } 10 string /IODevice resourceforall
       {(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

*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"

*%===== 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
*% Envelopes must be fed from the envelope feeder. Only envelopes may be fed
*% from the envelope feeder. Envelopes may not be transparent.
*UIConstraints: *PageSize Env10 *Duplex
*UIConstraints: *Duplex *PageSize Env10
*UIConstraints: *Duplex *MediaType Transparent
*UIConstraints: *MediaType Transparent *Duplex
*UIConstraints: *PageSize Legal *OutputBin Rear
*UIConstraints: *OutputBin Rear *PageSize Legal
*UIConstraints: *Option1 False *InputSlot Envelope
*UIConstraints: *InputSlot Envelope *Option1 False
*UIConstraints: *PageSize Env10 *InputSlot Upper
*UIConstraints: *InputSlot Upper *PageSize Env10
*UIConstraints: *PageSize Legal *InputSlot Envelope
*UIConstraints: *InputSlot Envelope *PageSize Legal
*UIConstraints: *PageSize Letter *InputSlot Envelope
*UIConstraints: *InputSlot Envelope *PageSize Letter
*UIConstraints: *PageSize Env10 *MediaType Transparent
*UIConstraints: *MediaType Transparent *PageSize Env10
*UIConstraints: *InputSlot Envelope *MediaType Transparent
*UIConstraints: *MediaType Transparent *InputSlot Envelope
```

```
*%=== 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 ===================
*% Print managers should use these entries to set paper size, unless there is
*% a specific reason to use PageRegion, such as with manual feed.
*OpenUI *PageSize: PickOne
*OrderDependency: 20 AnySetup *PageSize
*PageSize Letter: "(<<) cvx exec
  /PageSize [612 792] /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*PageSize Legal: "(<<) cvx exec
  /PageSize [612 1008] /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*PageSize Env10: "(<<) cvx exec /PageSize [297 684] /ImagingBBox null
  (>>) cvx exec setpagedevice"
*End
*DefaultPageSize: Letter
*?PageSize: "save
   currentpagedevice /PageSize get aload pop
   2 copy gt {exch} if (Unknown)
  (<<) cvx exec
  [612 792] (Letter)
  [612 1008] (Legal)
  [297 684] (Env10)
  (>>) cvx exec
   { 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: "(<<) cvx exec /PageSize [612 792] /ImagingBBox null
   (>>) cvx exec setpagedevice"
*End
*PageRegion Legal: "(<<) cvx exec /PageSize [612 1008] /ImagingBBox null
   (>>) cvx exec setpagedevice"
*End
*PageRegion Env10: "(<<) cvx exec /PageSize [297 684] /ImagingBBox null
   (>>) cvx exec setpagedevice"
*End
*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 Env10: "15 13 280 670"
*?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"
*PaperDimension Env10: "297 684"

*% On this device, the Upper tray is tray 0 and the Envelope tray is tray 1.
*OpenUI *InputSlot: PickOne
*OrderDependency: 15 AnySetup *InputSlot
*DefaultInputSlot: Upper
*InputSlot Upper: " mark {
   (<<) cvx exec
     /InputAttributes (<<) cvx exec /Priority [0] (>>) cvx exec
   (>>) cvx exec setpagedevice
  } stopped cleartomark "
*End

*InputSlot Envelope: " mark {
   (<<) cvx exec
     /InputAttributes (<<) cvx exec /Priority [1] (>>) cvx exec
   (>>) cvx exec setpagedevice
  } stopped cleartomark "
*End
```

```
*?InputSlot: "
save
   (<<) cvx exec
     /1 (Envelope)
     /0 (Upper)
   (>>) cvx exec
  currentpagedevice /InputAttributes get
  dup /Priority known
  { /Priority get 0 get (        ) cvs cvn get }
  {
    dup length 1 eq
    { {pop} forall (       ) cvs cvn get }
    { pop pop (Unknown) } ifelse
  } ifelse
  = flush
restore
"
*End
*CloseUI: *InputSlot

*OpenUI *MediaType: PickOne
*OrderDependency: 20 AnySetup *MediaType
*DefaultMediaType: Paper
*MediaType Transparent: "(<<) cvx exec /MediaType Transparent (>>) cvx exec setpagedevice"
*MediaType Paper: "(<<) cvx exec /MediaType Paper (>>) cvx exec setpagedevice"
*?MediaType: " save
   currentpagedevice /MediaType {get} stopped
   {pop pop (Unknown)} {dup null eq {pop (Unknown)} if} ifelse = flush restore "
*End
*CloseUI: *MediaType

*OpenUI *Duplex: PickOne
*OrderDependency: 30 AnySetup *Duplex
*DefaultDuplex: None
*Duplex DuplexTumble: "(<<) cvx exec /Duplex true /Tumble true (>>) cvx exec setpagedevice"
*Duplex DuplexNoTumble: "(<<) cvx exec /Duplex true /Tumble false (>>) cvx exec setpagedevice”
*Duplex None: "(<<) cvx exec /Duplex false /Tumble false (>>) cvx exec setpagedevice"
*?Duplex: "save currentpagedevice /Duplex get
   { currentpagedevice /Tumble get
       {(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
   { 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: "(<<) cvx exec /Separations true (>>) cvx exec setpagedevice"
*Separations False: "(<<) cvx exec /Separations false (>>) cvx exec 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
```

6.2 Level 2 Imagesetter

This PPD file describes a generic Level 2 roll-fed imagesetter. It supports Adobe's Accurate Screens technology and several resolutions. 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. While most imagesetters support custom page sizes, the custom page size section is not shown here because of its complexity; see section 6.3 for examples of custom page size code.

```
*PPD-Adobe: "4.3"
*FormatVersion: "4.3"
*FileVersion: "1.0"
*LanguageEncoding: ISOLatin1
*LanguageVersion: English
*Product: "(Acme Imagesetter)"
*PSVersion: "(2015.11) 7"
*Manufacturer: "Acme"
*ModelName: "Acme Imagesetter v.2015.11"
*ShortNickName: "Acme Imagesetter"
*NickName: "Acme Imagesetter v.2015.11"
*PCFileName: "ACIMAGE1.PPD"
*%======================================= Basic Capabilities ===============
*LanguageLevel: "2"
*ColorDevice: False
*DefaultColorSpace: Gray
*FreeVM: "8134935"
*FileSystem: True
*?FileSystem: "
  save false
     (%disk?%)
     { currentdevparams dup /Writeable known
       { /Writeable get {pop true} if }  { pop } ifelse
     } 10 string /IODevice resourceforall
     {(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

*%=== Resolution Information ===================
*OpenUI *Resolution/Choose Resolution: PickOne
*OrderDependency: 10 AnySetup *Resolution
*Resolution 600dpi: "(<<) cvx exec /HWResolution 600 (>>) cvx exec setpagedevice"
*Resolution 1200dpi: "(<<) cvx exec /HWResolution 1200 (>>) cvx exec setpagedevice"
*Resolution 2400dpi: "(<<) cvx exec /HWResolution 2400 (>>) cvx exec setpagedevice"
*DefaultResolution: 1200dpi
*?Resolution: " save
   currentpagedevice /HWResolution get
   0 get (          ) 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/Page Size: PickOne
*OrderDependency: 30 AnySetup *PageSize
*DefaultPageSize: Letter
*PageSize Letter: "(<<) cvx exec
  /PageSize [612 792]
  /ImagingBBox null (>>) cvx exec setpagedevice"
*End
```

```
*PageSize Legal: "(<<) cvx exec
  /PageSize [612 1008]
  /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*PageSize Tabloid: "(<<) cvx exec
  /PageSize [792 1224]
  /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*?PageSize: "save currentpagedevice /PageSize get aload pop
   2 copy gt {exch} if (Unknown)
  (<<) cvx exec
  [612 792] (Letter)
  [612 1008] (Legal)
  [792 1224] (Tabloid)
  (>>) cvx exec
   { 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. 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: "(<<) cvx exec
  /PageSize [612 792]
  /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*PageRegion Legal: "(<<) cvx exec
  /PageSize [612 1008]
  /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*PageRegion Tabloid: "(<<) cvx exec
  /PageSize [792 1224]
  /ImagingBBox null (>>) cvx exec setpagedevice"
*End
*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 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 Tabloid: "792 1224"

*% Only one input slot, but the entry is included to dictate the slot name
*% that appears in the user interface.
*OpenUI *InputSlot: PickOne
*OrderDependency: 30 AnySetup *InputSlot
*InputSlot Cassette: ""
*DefaultInputSlot: Cassette
*CloseUI: *InputSlot

*% === Imagesetter Information ==========================
*% Imagesetter features are grouped here.
*OpenGroup: Imagesetter

*OpenUI *MirrorPrint/Mirror Print: Boolean
*OrderDependency: 40 AnySetup *MirrorPrint
*MirrorPrint True: "(<<) cvx exec /MirrorPrint true (>>) cvx exec setpagedevice"
*MirrorPrint False: "(<<) cvx exec /MirrorPrint false (>>) cvx exec setpagedevice"
*DefaultMirrorPrint: False
*?MirrorPrint: " save currentpagedevice /MirrorPrint get
   {(True)} {(False)} ifelse = flush restore"
*End
*CloseUI: *MirrorPrint

*OpenUI *NegativePrint/Negative Print: Boolean
*OrderDependency: 40 AnySetup *NegativePrint
*NegativePrint True: "(<<) cvx exec /NegativePrint true (>>) cvx exec setpagedevice"
*NegativePrint False: "(<<) cvx exec /NegativePrint false (>>) cvx exec setpagedevice"
*DefaultNegativePrint: False
*?NegativePrint: "save currentpagedevice /NegativePrint get
   {(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
   { 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: Cassette not loaded"
*Status: "PrinterError: Film Unit Error"

*% Input Sources (format:%%[status:<stat>;source:<one of these>]%%)
*Source: "Localtalk"
*Source: "Parallel"
*Source: "Serial"
*Source: "SerialB"

*% 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"

*% end of PPD file for Acme Imagesetter
```

6.3 Examples of Custom Page Size Code

This section contains examples of custom page size entries for Level 1 and Level 2 drum and capstan recorders (roll-fed devices) and for Level 2 devices that accept both roll-fed and cut-sheet media. These examples are intended to help builders of PPD files with the construction of custom page size entries. They should be interpreted as guidelines, not requirements of this specification.

Because of the complexity of writing a custom page size entry, it is critical that you thoroughly test your *CustomPageSize code with requests for different page sizes, different offsets (if supported), and different orientations, preferably using several different print managers or applications. Such testing often highlights problems with the custom page size code in the PPD file, with a print manager's handling of custom page sizes, or even with the device's implementation of custom page sizes.

Minimums and maximums

When constructing your own custom page size entry, you would insert the device's values for *MaxMediaWidth, *MaxMediaHeight, *HWMargins (if present), and the minimum and maximum range values of the *ParamCustomPageSize entries. On some devices, requesting a width or height of zero will cause an error. Because of this, the values of the minimum Width and Height boundaries in the *ParamCustomPageSize entries should be set to small positive numbers, such as 100 points.

*LeadingEdge

*LeadingEdge is needed by print managers to give the user a way to request Transverse pages (on roll-fed media) or to tell the print manager how a cut-sheet device is configured so that the imageable area of the custom page size can be calculated correctly. It is important to include this keyword in the PPD file if *CustomPageSize is present.

For *LeadingEdge, list only the options that are available on the device. Most devices that accept only cut-sheet media will support either Short or Long, but not both. However, some cut-sheet devices will support both Short and Long, either in separate input trays or in an adjustable input tray. Most roll-fed devices support both Short and Long, because the page image can be rotated on the film or paper.

You'll need to write *NonUIConstraints that document which trays support which leading edge, such as these for a device whose Upper tray supports only long-edge feed and whose Lower tray supports only short-edge feed:

```
*LeadingEdge Short: ""
*LeadingEdge Long: ""
*DefaultLeadingEdge: Long
*NonUIConstraints: *InputSlot Upper *LeadingEdge Short
*NonUIConstraints: *LeadingEdge Short *InputSlot Upper
*NonUIConstraints: *InputSlot Lower *LeadingEdge Long
*NonUIConstraints: *LeadingEdge Long *InputSlot Lower
```

Typically, PreferLong will be available only on devices that accept roll-fed media. It should only appear in the PPD file as an option for *LeadingEdge if the device can be configured to calculate whether the page will fit on the roll in the long-edge feed direction and rotate the page to long-edge feed if it fits that way. PreferLong is typically not available on devices that accept **only** cut-sheet media, as the device has no way to rotate the physical page. If *HWMargins and *UseHWMargins are both present, then PreferLong will typically only be available when *UseHWMargins is False (when the device is operating in traditional roll-fed mode). If PreferLong is not available when *UseHWMargins is True, there should be a *NonUIConstraints entry to reflect this:

```
*NonUIConstraints: *UseHWMargins True *LeadingEdge PreferLong
*NonUIConstraints: *LeadingEdge PreferLong *UseHWMargins True
```

The *LeadingEdge option Forced is very rare. Forced will usually be available only on devices that accept cut-sheet media. If the device always rotates the page image in device space so that the long axis of the page image is parallel to the long axis of the physical page, then Forced is not available on the device. To find out whether Forced is available, try to print a short-edge feed image (Width < Height, Portrait orientation) on a long-edge feed page, or vice versa. You must first set up the device correctly, in "do what I say" mode, with the leading edge set up to be the opposite of what you will request from the print manager. If you do this with a full-page image, and Forced is available, the image should appear to be rotated 90 degrees and clipped. If you are unsure, do not include Forced as an option for *LeadingEdge in the PPD file.

Unknown is available for all devices, but it provides a print manager with no useful information and should be omitted if the device manufacturer (and PPD file builder) wants to force the user to choose the correct value (Short, Long, PreferLong, or Forced) for *LeadingEdge. If a user chooses Unknown, the print manager cannot correlate the top of the physical page with the *x* and *y* axes of the page image, so it cannot calculate the imageable area of the custom page size correctly. This can cause the print manager to falsely warn the user that the page image will be clipped, or it can cause actual clipping if the page image and the physical page are rotated relative to each other. For these reasons, including Unknown in the *LeadingEdge options list is discouraged. If Unknown must be listed as an option, using Unknown as the value of

*DefaultLeadingEdge is strongly discouraged. The value of *DefaultLeadingEdge should be the default leading edge of the default input slot; typically, Short, Long, or PreferLong.

Level 1 roll-fed devices

Since the *ParamCustomPageSize parameters for roll-fed devices are defined in terms of media feed direction, not fast scan/slow scan direction, writing the *CustomPageSize code and setting up the *ParamCustomPageSize entries can be confusing, and must be done with great care.

On Level 1 roll-fed devices that use the **setpageparams** operator in the *CustomPageSize code, HeightOffset is not used and will be discarded, as shown by **exch pop** in the first line of the *CustomPageSize code.

Example 1: *Entry for a Level 1 capstan recorder, or a Level 1 drum recorder where **setpageparams** has been redefined to emulate the output of a capstan recorder:*

```
*CenterRegistered: False
*LeadingEdge Short: ""
*LeadingEdge Long: ""
*DefaultLeadingEdge: Long
*NonUIOrderDependency: 20 AnySetup *CustomPageSize
*ParamCustomPageSize Width: 1 points 1 1008
*ParamCustomPageSize Height: 2 points 1 3000
*ParamCustomPageSize WidthOffset/Margins: 3 points 0 1007
*ParamCustomPageSize HeightOffset: 4 points 0 0
*ParamCustomPageSize Orientation: 5 int 0 1
*CustomPageSize True: "exch pop
  statusdict /setpageparams get exec"
*End
*MaxMediaWidth: "1008"
*?CurrentMediaWidth: "statusdict /mediawidth get exec = flush"
*MaxMediaHeight: "3000"
*?CurrentMediaHeight: "statusdict /medialength get exec = flush"
```

Example 2: *Entry for a Level 1 drum recorder, where* ***setpageparams*** *has not been redefined to emulate the output of a capstan recorder. Note the differences in the ordering of the parameters on the stack, and the differences in both the invocation and query code, compared to Example 1:*

```
*CenterRegistered: False
*LeadingEdge Short: ""
*LeadingEdge Long: ""
*LeadingEdge PreferLong: ""
*DefaultLeadingEdge: PreferLong
*NonUIOrderDependency: 20 AnySetup *CustomPageSize
*ParamCustomPageSize Width: 2 points 1 1152
*ParamCustomPageSize Height: 1 points 1 1584
*ParamCustomPageSize WidthOffset/Margins: 4 points 0 0
*ParamCustomPageSize HeightOffset: 3 points 0 1151
*ParamCustomPageSize Orientation: 5 int 0 1
*CustomPageSize True: "1 exch sub exch pop
  statusdict /setpageparams get exec"
*End
*MaxMediaWidth: "1152"
*?CurrentMediaWidth: "statusdict /medialength get exec = flush"
*MaxMediaHeight: "1584"
*?CurrentMediaHeight: "statusdict /mediawidth get exec = flush"
```

Level 2 devices

These examples of Level 2 *CustomPageSize invocation code first check the Orientation parameter to see if it is even or odd. An even Orientation parameter (0 or 2) means that the *x* axis will be parallel to the media feed direction, which means that Height must be mapped to the *x* axis. To perform this mapping, if Orientation is even, Width and Height are first checked for equality; if they are not equal, they are switched so that Height maps to *x* and Width maps to *y* in the **PageSize** array.

The code then examines the dimensions Width and Height to determine how to map the custom page size Orientation parameter to the **setpagedevice** key **Orientation**, using a choice of arrays of orientations. It then executes **setpagedevice** with the appropriate dictionary.

These examples are only appropriate for devices that support the **setpagedevice** key **PageOffset**; otherwise, this code must be written to discard the offset values provided by the user. The code used to obtain the maximum width and height is only appropriate for devices that support the **setpagedevice** key **OutputDevice** and a particular form of the **PageSize** value in the **OutputDevice** resource, as documented in the appropriate PostScript language supplement for the product.

This sample Level 2 code is more verbose than the equivalent Level 1 code because the mapping between the *ParamCustomPageSize parameters and the **setpagedevice** keys can be complicated. When writing your own code, consult the diagram of orientations in Figure 3, section 5.16 of this document. If the value of Orientation produces different positions of the pages on the media than described in these examples, you can still use the examples as a starting point and change the numbers in the arrays to correctly map the custom page size parameter Orientation to the **setpagedevice** key **Orientation** in the invocation code.

Example 3: *Entry for a Level 2 capstan recorder that accepts only roll-fed media. This example assumes that a portrait page with* ***Orientation*** *equal to* ***0*** *in the* ***currentpagedevice*** *dictionary results in the same positioning of the page on the media as when the *ParamCustomPageSize parameter Orientation is 0 and Width is greater than the Height. Note the order of the parameters on the stack dictated by *ParamCustomPageSize.*

```
*CenterRegistered: False
*DefaultLeadingEdge: PreferLong
*LeadingEdge Long: ""
*LeadingEdge PreferLong: ""
*LeadingEdge Short: ""
*ParamCustomPageSize Width: 3 points 100 1008
*ParamCustomPageSize Height: 4 points 100 3000
*ParamCustomPageSize WidthOffset/Width Margin: 1 points 0 908
*ParamCustomPageSize HeightOffset/Height Margin: 2 points 0 2900
*ParamCustomPageSize Orientation: 5 int 0 3
*NonUIOrderDependency: 30 AnySetup *CustomPageSize
*CustomPageSize True: "
3 copy 2 mod 0 eq {
  2 copy eq {
    1 add
  }{
    5 -2 roll exch 5 2 roll
  } ifelse
} if
[0 0 2 2] [3 1 1 3]
4 2 roll lt {exch} if pop
exch get
(<<) cvx exec
  /Orientation 3 -1 roll
  /PageSize [ 7 -2 roll ]
  /PageOffset [ 9 -2 roll ]
  /ImagingBBox null
(>>) cvx exec setpagedevice"
*End
*MaxMediaWidth: "1008"
*?CurrentMediaWidth: "
  currentpagedevice /OutputDevice get
  /OutputDevice findresource
  /PageSize get 0 get 2 get = flush
"
*End
*MaxMediaHeight: "3000"
*?CurrentMediaHeight: "
  currentpagedevice /OutputDevice get
  /OutputDevice findresource
  /PageSize get 0 get 3 get = flush
"
*End
```

Example 4: *The code for a roll-fed drum recorder may be slightly different from the code for a capstan recorder. This example assumes that the* **setpagedevice** *key* **Orientation** *on the drum recorder produces the same results as the capstan device in Example 3. This code is similar to Example 3, except that it first calculates a new value for the width offset by subtracting the custom page size parameters* Width *and the* WidthOffset *from the maximum width. This is necessary because the fast scan direction (relative to media feed direction) of a drum recorder is different from that of a capstan recorder. Note that the order of the parameters on the stack is different from that of Example 3.*

```
*CenterRegistered: False
*DefaultLeadingEdge: PreferLong
*LeadingEdge Long: ""
*LeadingEdge PreferLong: ""
*LeadingEdge Short: ""
*ParamCustomPageSize Width: 1 points 100 1152
*ParamCustomPageSize Height: 2 points 100 1584
*ParamCustomPageSize WidthOffset/Width Margin: 5 points 0 1052
*ParamCustomPageSize HeightOffset/Height Margin: 4 points 0 1484
*ParamCustomPageSize Orientation: 3 int 0 3
*NonUIOrderDependency: 30 AnySetup *CustomPageSize
*CustomPageSize True: "
   currentpagedevice /OutputDevice get /OutputDevice findresource
   /PageSize get 0 get 3 get
   5 index sub exch sub 5 2 roll
   3 copy 2 mod 0 eq {
    2 copy eq {
    1 add
    }{
    5 -2 roll exch 5 2 roll
    } ifelse
   } if  %
   [0 0 2 2] [3 1 1 3]
      4 2 roll lt {exch} if pop exch get
   (<<) cvx exec
    /Orientation 3 -1 roll
    /PageSize [ 7 -2 roll ]
    /PageOffset [ 9 -2 roll ]
    /ImagingBBox null
   (>>) cvx exec setpagedevice
"
*End
*MaxMediaWidth: "1152"
*?CurrentMediaWidth: "
  currentpagedevice /OutputDevice get
  /OutputDevice findresource
  /PageSize get 0 get 3 get = flush
"
*End
*MaxMediaHeight: "1584"
*?CurrentMediaHeight: "
  currentpagedevice /OutputDevice get
  /OutputDevice findresource
  /PageSize get 0 get 2 get = flush
"
*End
```

Cut-sheet media

If the roll-fed capstan recorder depicted in Example 3 also accepted cut-sheet media, additional entries would be required in the PPD file, as shown in Example 5. Example 6 illustrates a device that, unlike all the other examples in this section, accepts **only** cut-sheet media, perhaps in an input tray whose sides can be adjusted to accommodate different page sizes.

Example 5: *In this example, cut-sheet media is accepted only through the manual feed slot, fed short-edge first, and the hardware imposes margins of 1 inch at the top and bottom and 1/2 inch on the sides. These entries would appear in addition to the entries shown in Example 3.*

```
*HWMargins: 72 36 72 36
*UseHWMargins True: ""
*UseHWMargins False: ""
*DefaultUseHWMargins: False
*NonUIConstraints: *InputSlot Cassette *UseHWMargins True
*NonUIConstraints: *UseHWMargins True *InputSlot Cassette
*NonUIConstraints: *ManualFeed True *UseHWMargins False
*NonUIConstraints: *UseHWMargins False *ManualFeed True
*NonUIConstraints: *InputSlot Cassette *ManualFeed True
*NonUIConstraints: *ManualFeed True *InputSlot Cassette
*NonUIConstraints: *ManualFeed True *LeadingEdge PreferLong
*NonUIConstraints: *ManualFeed True *LeadingEdge Long
*NonUIConstraints: *LeadingEdge Long *ManualFeed True
*NonUIConstraints: *LeadingEdge PreferLong *ManualFeed True
```

WidthOffset and HeightOffset on cut-sheet devices

Most devices that accept only cut-sheet media do not support the concept of offsetting the image in a particular direction. On such devices, WidthOffset and HeightOffset should be discarded by the *CustomPageSize code. Level 2 devices that do support offsetting will do so with one of the following **setpagedevice** keys:

- **Margins,** which should not be used for this purpose. **Margins**, which is measured in device units, is intended to be used to compensate for mechanical misadjustments in the device, not to position output on the page. Do not use it in *CustomPageSize code.

- **PageOffset,** which is intended for this purpose. WidthOffset, HeightOffset, and PageOffset are all defined in “points” (1/72 of an inch), so no conversion is necessary. However, the direction of offset for **PageOffset** is in device space, which may cause problems for a rotated page. Test the custom page size code carefully with different combinations of Orientation, WidthOffset, and HeightOffset. In each case, WidthOffset should remain relative to Width, and HeightOffset should remain relative to Height. If this does not happen, you will have to amend the *CustomPageSize code so that it manipulates the offset values to compensate for Orientation before passing them to **PageOffset**.

- **ImageShift** is defined in default user space, so the *CustomPageSize code can pass the values of WidthOffset and HeightOffset directly into **ImageShift**. However, if the device also performs duplex printing on custom page sizes, this combination should be carefully tested with WidthOffset and HeightOffset to make sure the image is offset in the proper direction on both sides of the page, as **ImageShift** was designed to shift the image differently on the front and back sides of a page. In this case, **PageOffset** may work better.

If the device does not support either of the **setpagedevice** keys **PageOffset** or **ImageShift**, or if the manufacturer does not want image offsetting to be supported in custom page sizes on the device, change the *min* and *max* range values for WidthOffset and HeightOffset to 0:

```
*ParamCustomPageSize WidthOffset: 3 points 0 0
*ParamCustomPageSize HeightOffset: 4 points 0 0
```

and rewrite the *CustomPageSize code to discard the values of WidthOffset and HeightOffset. Be careful to keep the dictionary mark on the stack.

Example 6: *This is a sample custom page size entry for a PostScript Level 2 device that accepts only cut-sheet media, supports custom page sizes fed short-edge first only through the manual feed slot, and does not support offsetting or the* ***setpagedevice*** *key* ***Orientation****. The *CustomPageSize code first discards the values of HeightOffset and WidthOffset by popping them off the stack. (Note the order of the parameters.) Next, Orientation is checked (and discarded, since it is not supported by the device); if it is even, the values of Width and Height are exchanged so that they will be in the correct order for the requested landscape or portrait page. This code assumes that Width, Height, and Orientation have been calculated as described in Table 2 in section 5.16 and that all values were put on the stack in the correct order.*

```
*HWMargins: 72 36 72 36
*LeadingEdge Short: ""
*DefaultLeadingEdge: Short
*NonUIConstraints: *InputSlot Upper *ManualFeed True
*NonUIConstraints: *ManualFeed True *InputSlot Upper
*NonUIConstraints: *ManualFeed False *CustomPageSize True
*NonUIConstraints: *CustomPageSize True *ManualFeed False
*MaxMediaWidth: 792
*MaxMediaHeight: 1008
*CustomPageSize True: "
   pop pop
   2 mod 0 eq {exch} if
   (<<) cvx exec
   /PageSize [ 5 -2 roll ]
  /ImagingBBox null
   (>>) cvx exec setpagedevice
"
*End
*ParamCustomPageSize Width: 1 points 100 1008
*ParamCustomPageSize Height: 2 points 100 1008
*ParamCustomPageSize WidthOffset: 4 points 0 0
*ParamCustomPageSize HeightOffset: 5 points 0 0
*ParamCustomPageSize Orientation: 3 int 0 3
```

*?CurrentMediaHeight, *?CurrentMediaWidth, and *CenterRegistered are not usually supported by devices that accept only cut-sheet media, so they are omitted from this example. *UseHWMargins is omitted because this device can only print within the hardware-imposed margins (this is true for most cut-sheet-only devices), so there is no choice to be made about using *HWMargins (it must be used).

7 PPD File Summary

This section is intended for builders of PPD files. It provides a brief summary of things to check for when you're done building your PPD file. Checking off all the items on this list does not guarantee that you've built a perfect PPD file, but at least you will have avoided some common errors and omissions.

- Using the *PPD File Format Specification* and the product addendum as a guide, first make sure that the PPD file contains **all** of the keywords from the specification that are relevant to the device.

- Using the product addendum, make sure that the PPD file does not contain any entries or statements for features that are **not** supported by the device.

- Make sure the required keywords are present and that they are correct according to their descriptions in section 5.

- Make sure that *PCFileName, *ModelName, *NickName, and *ShortNickName are unique across all PPD files, and that *ShortNickName is < 32 characters and occurs before *NickName.

- If there is an ICC color profile for the device: Check *ModelName, *NickName, and *ShortNickName against the value of icHeader.model in the ICC color profile, and check the first four characters of these strings against icHeader.manufacturer.

- Make sure the *Manufacturer string is the same as in all other PPD files from this manufacturer. Check its value against the icHeader.manufacturer tag in any ICC color profiles for that manufacturer.

- Make sure that the *PCFileName follows Adobe naming standards.

- Any main keywords created by the manufacturer must start with the manufacturer's assigned prefix (see *Appendix D*).

- Check the *UIConstraints entries and *NonUIConstraints entries to make sure that all necessary constraints exist, that there are no unnecessary constraints, that constraints that need to be reversed have been reversed, and that no constraints are reversed unnecessarily.

- Make sure that the four required media handling keywords (*PageSize, *PageRegion, *ImageableArea, *PaperDimension) are present for each page size. Make sure any translation strings on option keywords are consistent across all media handling keywords.

- The page size option keyword qualifier Transverse is often misused. Check for the following:

  - 1. Transverse sizes should not appear in the PPD file of a cut-sheet device that supports either short-edge-feed or long-edge-feed but not both. If a device supports only one feed direction, there is no reason to have Transverse sizes listed in the PPD. Transverse is used to designate a page size that is **fed differently** from other page sizes on the device.

  - 2. If a cut-sheet device does support 2 media feed directions and they are selectable via PostScript code, the *PageSize code for equivalent Transverse and non-Transverse sizes (for example, Letter and Letter.Transverse) should request the same page size (with the dimensions in the same order, to preserve the correct image orientation on the page), but should do whatever it takes to ensure that the page is fed from the correct tray. This might mean that the code requests a different page orientation (if the **setpagedevice** key /**Orientation** is supported by the device) or a different input slot, as shown here:

```
*PageSize Letter: ".../PageSize [612 792]
   /MediaPosition 1..."
*End
*PageSize Letter.Transverse: ".../PageSize [612 792]
   /MediaPosition 0..."
*End
```

    The whole point of Transverse is for the user to say "this device usually feeds Letter from the short-edge-feed input tray, but this time I want a Letter page from the long-edge-feed tray". The physical size is the same, but the feed direction is different, and code needs to be sent to invoke that difference. However, the *PageRegion code fragments for the equivalent Transverse and non-Transverse sizes will usually be identical to each other, because *PageRegion should not select an input tray; it should only request the page dimensions,which are the same for the Transverse and non-Transverse size.

  - 3. On a roll-fed device, the *PageSize code for Transverse pages will request a different /**Orientation** than the *PageSize code for the equivalent non-Transverse size. For example:

```
*PageSize Letter: ".../PageSize [612 792]
   /Orientation 1..."
*End
*PageSize Letter.Transverse: ".../PageSize [612 792]
   /Orientation 0..."
*End
```

Again, the page size dimensions should be in the same order, at least for most roll-fed devices. The *PageSize and *PageRegion code fragments will usually be identical, since on a roll-fed device the *PageSize code does not usually invoke an input slot, so it performs the same function as the *PageRegion code.

- 4. The page size dimensions should be in the same order in all relevant keywords (*PageSize, *PageRegion, *ImageableArea, *PaperDimension) for equivalent Transverse and non-Transverse sizes.

- 5. If the intent is to produce a landscape page, you should use Rotated or R as the suffix (for example, A4Rotated or LetterR) instead of Transverse. Most print managers perform landscape rotation internally, so there's really no need for special landscape page sizes in a PPD file, unless it will be used in an environment where the print manager does not provide this service. If landscape page sizes are included in the PPD file, their dimensions in all relevant keywords should be in reversed from the dimensions of the equivalent non-Transverse size. For example:

```
*PaperDimension LetterRotated: "792 612"
*PaperDimension Letter: "612 792"
```

- Make sure that the imageable area for each page size is within its bounding box as defined by *PaperDimension for that page size.

- Make sure the *?ImageableArea query ends with the following code, as in the examples in section 6:

```
repeat ( ) = flush
```

  rather than the old form of code, which was

```
repeat flush
```

  The first example ensures that the string returned from the query terminates with a newline, which is a new requirement of the 4.3 specification.

- Make sure that all user-selectable features are surrounded by *OpenUI/ *CloseUI pairs as needed, and that each entry is complete (contains a *Default keyword, *OrderDependency statement, main keyword with options and code to invoke the feature, and query code if a query keyword exists for that feature and if query code can be written).

- Make sure that the strings returned by query code do **not** include **any** translation strings; they should return only option names, and they must be terminated by a newline (usually accomplished by ending the code with the sequence **= flush**).

- Make sure the device's error and status messages have been included.

- Make sure that color separation information is present for each resolution.

- Most critical: Make sure that every piece of PostScript code in the PPD file has been thoroughly tested against the device, with all variables, such as invoking different input slots and page sizes.

- Test the PPD file with different print managers and applications. Make sure that all features that are surrounded by *OpenUI/*CloseUI in the PPD file are selectable from the print manager. Test the functionality of the print manager and PPD file by printing a sample document, using the different PPD file options. For example, select Duplex and Lower Tray and make sure that document is printed from the lower tray and is printed on both sides of the page. Test the functionality of **all** the features in the PPD file. Test the PPD file and print manager by printing from more than one application.

7.1 PPD Files for Kanji Products

If you are building a PPD file for a Kanji (Japanese font) product, you should be aware of the following guidelines, which have evolved over time and are now an accepted part of the PPD file building process. These are not requirements of this specification, and failure to follow these guidelines should not be considered an error in the PPD file. They are recommendations, intended to provide some uniformity in PPD files.

- *ModelName should include the word “Kanji”. For example:

  ```
  *ModelName: "ACME Maxi-Print Kanji"
  ```

- *NickName should include the word “Kanji” as well as the PostScript interpreter version number. For example:

  ```
  *NickName: "ACME Maxi-Print Kanji v2013.114
  ```

- *ShortNickName should include the word “Kanji”. If you have to shorten the name to squeeze it into 31 characters, shorten another part of the name. Leave the word “Kanji” to differentiate this product from any equivalent Roman product.

- *PCFileName: The 8.3 format name should include a “J” in the file name, usually near the end of the name. For example: *xxxxxJ_1.PPD* or *xxxxxxJ1.PPD.* The first two characters must be the assigned manufacturer prefix.

- *PageSize, *PageRegion, *ImageableArea, *PaperDimension: Make sure that any additional page sizes available on the Kanji product are included, and that any page sizes that are not available on the Kanji product are deleted.

- *Font: Make sure that all Kanji fonts built into the product are included in the list with the correct Character Set, Font Encoding, and Status fields filled in. This includes fonts on a separate hard disk **if** the hard disk always ships with the product. If the hard disk is not part of the minimal configuration for the product, you should create one PPD file for the device in its minimal configuration, and a second PPD file that represents the product with the optional hard disk attached.

- If the Kanji product comes with more memory than the Roman product, you will need to make the appropriate changes to *FreeVM, *InstalledMemory, and *VMOption.

- If you are translating strings in the PPD file into Japanese, you will need to change *LanguageVersion and *LanguageEncoding to the appropriate value. If the PPD file is to be a cross-platform (portable) file, you must use the hexadecimal notation for 8-bit byte codes in translation strings. If the file is platform-specific (for example, Macintosh-only or Windows-only), you may use 8-bit byte codes directly in the translation strings. You may also directly translate the values of the following keywords into 8-bit byte codes, without using translation strings: *ModelName, *ShortNickName, *NickName, *Manufacturer, *PCFileName, *Include.

Appendix A: Keyword Categories

A.1 UI Keywords

This section provides a list of keywords that are typically bracketed by the *OpenUI/*CloseUI keywords in PPD files. 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.

*AdvanceMedia
*BindColor
*BindEdge
*BindType
*BindWhen
*BitsPerPixel
*BlackSubstitution
*Booklet
*Collate
*ColorModel
*CutMedia
*Duplex
*ExitJamRecovery
*FoldType
*FoldWhen
*InputSlot
*InstalledMemory
*Jog
*ManualFeed
*MediaColor
*MediaType
*MediaWeight
*MirrorPrint
*NegativePrint
*OutputBin
*OutputMode
*OutputOrder
*PageSize
*PageRegion
*Separations
*Signature
*Slipsheet
*Smoothing
*Sorter
*StapleLocation
*StapleOrientation
*StapleWhen
*StapleX
*StapleY
*TraySwitch

A.2 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 that do not have option keywords to distinguish one instance from another, yet all instances are relevant, so all occurrences of this main keyword and its associated unique values should be recorded by the PPD file parser.

```
*HalftoneName
*Include
*Message
*NonUIConstraints
*NonUIOrderDependency
*OrderDependency
*PageDeviceName
*PrinterError
*Product
*PSVersion
*QueryOrderDependency
*RenderingIntent
*Source
*Status
*UIConstraints
```

Appendix B: Registered *mediaOption* Keywords

The tables in this section contain the option keywords currently registered for *mediaOption*, which designates a given page size on a device. *Table B.1* is sorted by the *mediaOption* name. Given the name of a page size, the table provides the dimensions and any additional information about that size.

Table B.2 is sorted by size. This is useful for a person building a PPD file, as the dimensions of a page size available on a device can be looked up in the table and it can be determined if there is already a *mediaOption* defined for that size.

Only the most common page sizes are specified here. A device manufacturer is free to list a new size in a PPD file for a new device. However, care should be taken to avoid duplicating the semantics of an already-registered option keyword. Also, when creating a new option keyword, the capitalization conventions shown in the tables should be followed as much as possible; that is, the first letters of logical words should be capitalized.

Note *Builders of PPD files should always use existing mediaOption names where possible, because some applications and users have come to rely on these standard names for page sizes. Also, keeping paper size names consistent reduces user confusion. Builders of PPD files should define new mediaOptions* ***only*** *when there is no existing mediaOption that describes a certain page size. See section 5.13 for more information on mediaOption, and section 5.1 for information on defining new option keywords.*

B.1 Components of *mediaOption* Keywords

Any *mediaOption* keyword can be qualified by another string that indicates a slightly distinct treatment of the media size. A qualifier is appended with a period, like this:

```
Letter.Transverse
```

Any *mediaOption* keyword can have a serialization qualifier, 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 qualifiers can be combined with other qualifiers. For example, Letter.Transverse.1

Certain qualifiers and substrings in *mediaOption* names have special meaning. The most common components of a *mediaOption* name are defined here, primarily so that builders of PPD files can construct meaningful names for their page sizes:

- As of the 4.3 specification, the prefix Env denotes an envelope page size. The following sizes had the prefix Env added to reflect this change:C0, C1, C2, C3, C4, C5, C6, C7, DL, and Monarch. Additionally, Comm10 was changed to Env10. However, these size names were **not** changed in existing PPD files. While the old names are still valid, PPD files built to conform to the 4.3 specification should use EnvC0, EnvC1, etc. rather than C0, C1, etc. This enables a print manager to recognize envelope sizes and group them together. Translations strings can be used to control the *mediaOption* name that is displayed to the user.

- The size Executive varies by as much as 1/2 inch across devices. Most devices offer only one version of Executive. However, if a device offers more than one size of Executive, these sizes can be differentiated by a serialization extension and a translation string that denotes the exact size. For example, a PPD file for a Level 1 device might have:

```
*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"
```

- Envelope describes envelopes that have no standard name. This keyword can be qualified by an *x* and *y* dimension (specified in PostScript default units), in the order **x.y**, where *x* is perpendicular to the feed direction and *y* is parallel to the feed direction. In the 3.0 specification, all envelope sizes were specified in the format Envelope.x.y, usually with a translation string for clarity. In later versions, envelopes are simply another size of media, and most envelope sizes are listed by their common names, but the 3.0 format is still valid and is useful for envelopes without common names.

- Extra in a page size name or as a qualifier designates a page size slightly larger than the corresponding standard size, such as A4Extra. The purpose is to allow for bleeds and crop marks to be printed in the margins. However, the increase in size is somewhat variable across devices. Typically, an Extra size is 0.69 to 1 inch larger than its corresponding standard size.

- MaxPage is a special mediaOption that denotes the largest page size available on a given device. The dimensions of this size will vary widely across devices. MaxPage provides a convenient way for the user to select the largest available page size.

- Rotated in a page size name designates a page size whose image is rotated relative to the corresponding standard size. For most page sizes, this produces a "landscape" image on the page. For example, on a LetterRotated page, the x dimension is longer than the y dimension. This functionality is usually provided by the print manager on the host, so the use of Rotated page sizes in a PPD file is usually redundant and should be discouraged.

- Small in a page size name or as a qualifier usually denotes a page that is the same physical size as the regular named page, but with an imageable area that is typically 10 to 20 points smaller.

- The Transverse qualifier 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 usually means "long edge feed." That is, a Transverse size 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.

  This 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*. 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. On older devices, the printing speed when using the image operator might be different for a page that is fed transversely.

  Because the orientation of the image on the page is the same, the dimensions for a Transverse and non-Transverse page are the same. For example:

  ```
  *PaperDimension Letter: "612 792"
  *PaperDimension Letter.Transverse: "612 792"
  ```

Page sizes with the qualifier Transverse should only appear in the PPD file of a roll-fed device (such as an imagesetter) that supports requesting a specific orientation so that pages can be either short-edge feed or long-edge feed, or in the PPD file of a cut-sheet device that has both short-edge-feed and long-edge-feed input trays. If a cut-sheet device supports either short-edge-feed or long-edge-feed, but not both, there is no need to differentiate Transverse pages from non-Transverse pages. Transverse is only useful if there is a need to choose between two input slots with different feed directions.

If a page size has both a Transverse and non-Transverse version in a PPD file, their *PageSize code fragments should be different. For example, in a PPD file for a Level 2 device, the *PageSize code for Letter.Transverse should request a different /**Orientation** from **setpagedevice** than the code for Letter, or it should set up a different input slot. Otherwise, including both the Transverse and non-Transverse size is pointless, if they produce exactly the same result. See section 7 for more advice and examples of putting Transverse sizes in a PPD file.

B.2 *mediaOption* Name Tables

The dimensions given in the tables are in PostScript default units. They refer to the actual physical dimensions of the page, not the imageable region, unless otherwise specified. All sizes are given with the *x* dimension first, followed by the *y* dimension. The units in which a page size was originally defined appear in boldface type. The dimensions in other units are provided strictly for comparison, and are approximate due to rounding.

The letters in the **Notes** column have the following meanings:

- **I**—size is defined by ISO standards
- **J**—size is defined by JIS standards
- **S**—imageable area is smaller than the imageable area of the corresponding standard size, typically by 10 to 20 points
- **N**—see section B.1 for information about this size
- **V**—size varies across devices, typically by up to 1/2 inch
- **E**—this is an envelope size

Microsoft has defined programming identifiers (#define's in C language programs) for many of the common mediaOption keywords, for use in Windows applications. This enables a print manager to match a request for a specific page size, in the form of a #define, to a mediaOption name.The **#define** column contains those identifiers where appropriate.

Table B.1 *MediaOptions Sorted By Name*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
10x11	720 x 792	254 x 279.4	**10 x 11**		DMPAPER_10X11
10x13	720 x 936	254 x 330.2	**10 x 13**		
10x14	720 x 1008	254 x 355.6	**10 x 14**		DMPAPER_10X14
12x11	864 x 792	304.8 x 279.4	**12 x 11**		DMPAPER_12X11
15x11	1080 x 792	381 x 279.4	**15 x 11**		DMPAPER_15X11
7x9	504 x 648	177.8 x 228.6	**7 x 9**		
8x10	576 x 720	203.2 x 254	**8 x 10**		
9x11	648 x 792	228.6 x 279.4	**9 x 11**		DMPAPER_9X11
9x12	648 x 864	228.6 x 304.8	**9 x 12**		
A0	2384 x 3370	**841 x 1189**	33.11 x 46.81	I, J	
A1	1684 x 2384	**594 x 841**	23.39 x 33.11	I, J	
A2	1191 x 1684	**420 x 594**	16.54 x 23.39	I, J	DMPAPER_A2
A3	842 x 1191	**297 x 420**	11.69 x 16.54	I, J	DMPAPER_A3
A3.Transverse	842 x 1191	**297 x 420**	11.69 x 16.54		DMPAPER_A3_TRANSVERSE
A3Extra	913 x 1262	**322 x 445**	12.67 x 17.52	N, V	DMPAPER_A3_EXTRA
A3Extra.Transverse	913 x 1262	**322 x 445**	12.67 x 17.52	N, V	DMPAPER_A3_EXTRA_TRANSVERSE
A3Rotated	1191 x 842	**420 x 297**	16.54 x 11.69	N	DMPAPER_A3_ROTATED
A4	595 x 842	**210 x 297**	8.27 x 11.69	I, J	DMPAPER_A4
A4.Transverse	595 x 842	**210 x 297**	8.27 x 11.69		DMPAPER_A4_TRANSVERSE
A4Extra	667 x 914	235.5 x 322.3	**9.27 x 12.69**	N, V	DMPAPER_A4_EXTRA
A4Plus	595 x 936	**210 x 330**	8.27 x 13		DMPAPER_A4_PLUS
A4Rotated	842 x 595	**297 x 210**	11.69 x 8.27	N	DMPAPER_A4_ROTATED
A4Small	595 x 842	**210 x 297**	8.27 x 11.69	S	DMPAPER_A4SMALL
A5	420 x 595	**148 x 210**	5.83 x 8.27	I, J	DMPAPER_A5
A5.Transverse	420 x 595	**148 x 210**	5.83 x 8.27		DMPAPER_A5_TRANSVERSE
A5Extra	492 x 668	**174 x 235**	6.85 x 9.25	N, V	DMPAPER_A5_EXTRA
A5Rotated	595 x 420	**210 x 148**	8.27 x 5.83	N	DMPAPER_A5_ROTATED
A6	297 x 420	**105 x 148**	4.13 x 5.83	I, J	DMPAPER_A6
A6Rotated	420 x 297	**148 x 105**	5.83 x 4.13	N	DMPAPER_A6_ROTATED

Table B.1 *MediaOptions Sorted By Name (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
A7	210 x 297	**74 x 105**	2.91 x 4.13	I, J	
A8	148 x 210	**52 x 74**	2.05 x 2.91	I, J	
A9	105 x 148	**37 x 52**	1.46 x 2.05	I, J	
A10	73 x 105	**26 x 37**	1.02 x 1.46	I, J	
AnsiC	1224 x 1584	431.8 x 558.8	**17 x 22**		
AnsiD	1584 x 2448	558.8 x 863.6	**22 x 34**		
AnsiE	2448 x 3168	863.6 x 1118	**34 x 44**		
ARCHA	648 x 864	228.6 x 304.8	**9 x 12**		
ARCHB	864 x 1296	304.8 x 457.2	**12 x 18**		
ARCHC	1296 x 1728	457.2 x 609.6	**18 x 24**		DMPAPER_CSHEET
ARCHD	1728 x 2592	609.6 x 914.4	**24 x 36**		DMPAPER_DSHEET
ARCHE	2592 x 3456	914.4 x 1219	**36 x 48**		DMPAPER_ESHEET
B0	2920 x 4127	**1030 x 1456**	40.55 x 57.32	J	
B1	2064 x 2920	**728 x 1030**	28.66 x 40.55	J	
B2	1460 x 2064	**515 x 728**	20.28 x 28.66	J	
B3	1032 x 1460	**364 x 515**	14.33 x 20.28	J	
B4	729 x 1032	**257 x 364**	10.12 x 14.33	J	DMPAPER_B4
B4Rotated	1032 x 729	**364 x 257**	14.33 x 10.12	N	DMPAPER_B4_JIS_ROTATED
B5	516 x 729	**182 x 257**	7.17 x 10.12	J	DMPAPER_B5
B5.Transverse	516 x 729	**182 x 257**	7.17 x 10.12		DMPAPER_B5_TRANSVERSE
B5Rotated	729 x 516	**257 x 182**	10.12 x 7.17	N	DMPAPER_B5_JIS_ROTATED
B6	363 x 516	**128 x 182**	5.04 x 7.17	J	DMPAPER_B6_JIS
B6Rotated	516 x 363	**182 x 128**	7.17 x 5.04	N	DMPAPER_B6_JIS_ROTATED
B7	258 x 363	**91 x 128**	3.58 x 5.04	J	
B8	181 x 258	**64 x 91**	2.52 x 3.58	J	
B9	127 x 181	**45 x 64**	1.77 x 2.52	J	
B10	91 x 127	**32 x 45**	1.26 x 1.77	J	
C4 (use **EnvC4**)	649 x 918	**229 x 324**	9.02 x 12.75	I, E, N	DMPAPER_ENV_C4
C5 (use **EnvC5**)	459 x 649	**162 x 229**	6.38 x 9.02	I, E, N	DMPAPER_ENV_C5
C6 (use **EnvC6**)	323 x 459	**114 x 162**	4.49 x 6.38	I, E, N	DMPAPER_ENV_C6

Table B.1 *MediaOptions Sorted By Name (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
Comm10 (use **Env10**)	297 x 684	104.8 x 241.3	**4.125 x 9.5**	E, N	DMPAPER_ENV_10
DL (use **EnvDL**)	312 x 624	**110 x 220**	4.33 x 8.66	I, E, N	DMPAPER_ENV_DL
DoublePostcard	567 x 419.5	**200 x 148**	7.87 x 5.83		DMPAPER_DBL_JAPANESE_POSTCARD
DoublePostcardRotated	419.5 x 567	**148 x 200**	5.83 x 7.87	N	DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED
Env9	279 x 639	98.4 x 225.4	**3.875 x 8.875**	E	DMPAPER_ENV_9
Env10	297 x 684	104.8 x 241.3	**4.125 x 9.5**	E	DMPAPER_ENV_10
Env11	324 x 747	114.3 x 263.5	**4.5 x 10.375**	E	DMPAPER_ENV_11
Env12	342 x 792	120.7 x 279.4	**4.75 x 11**	E	DMPAPER_ENV_12
Env14	360 x 828	127 x 292.1	**5 x 11.5**	E	DMPAPER_ENV_14
EnvC0	2599 x 3676	**917 x 1297**	36.10 x 51.06	I, E	
EnvC1	1837 x 2599	**648 x 917**	25.51 x 36.10	I, E	
EnvC2	1298 x 1837	**458 x 648**	18.03 x 25.51	I, E	
EnvC3	918 x 1296	**324 x 458**	12.75 x 18.03	I, E	DMPAPER_ENV_C3
EnvC4	649 x 918	**229 x 324**	9.02 x 12.75	I, E	DMPAPER_ENV_C4
EnvC5	459 x 649	**162 x 229**	6.38 x 9.02	I, E	DMPAPER_ENV_C5
EnvC6	323 x 459	**114 x 162**	4.49 x 6.38	I, E	DMPAPER_ENV_C6
EnvC65	324 x 648	**114 x 229**	4.5 x 9	E	DMPAPER_ENV_C65
EnvC7	230 x 323	**81 x 114**	3.19 x 4.49	I, E	
EnvChou3	340 x 666	**120 x 235**	4.72 x 9.25	E	DMPAPER_JENV_CHOU3
EnvChou3Rotated	666 x 340	**235 x 120**	9.25 x 4.72	E, N	DMPAPER_JENV_CHOU3_ROTATED
EnvChou4	255 x 581	**90 x 205**	3.54 x 8	E	DMPAPER_JENV_CHOU4
EnvChou4Rotated	581 x 255	**205 x 90**	8 x 3.54	E, N	DMPAPER_JENV_CHOU4_ROTATED
EnvDL	312 x 624	**110 x 220**	4.33 x 8.66	I, E	DMPAPER_ENV_DL
EnvInvite	624 x 624	**220 x 220**	8.66 x 8.66	E	DMPAPER_ENV_INVITE
EnvISOB4	708 x 1001	**250 x 353**	9.84 x 13.9	E	DMPAPER_ENV_B4
EnvISOB5	499 x 709	**176 x 250**	6.9 x 9.8	E	DMPAPER_ENV_B5
EnvISOB6	499 x 354	**176 x 125**	6.9 x 4.9	E	DMPAPER_ENV_B6
EnvItalian	312 x 652	**110 x 230**	4.33 x 9	E	DMPAPER_ENV_ITALY
EnvKaku2	680 x 941	**240 x 332**	9.45 x 13	E	DMPAPER_JENV_KAKU2
EnvKaku2Rotated	941 x 680	**332 x 240**	13 x 9.45	E, N	DMPAPER_JENV_KAKU2_ROTATED

Table B.1 *MediaOptions Sorted By Name (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
EnvKaku3	612 x 785	**216 x 277**	8.5 x 10.9	E	DMPAPER_JENV_KAKU3
EnvKaku3Rotated	785 x 612	**277 x 216**	10.9 x 8.5	E, N	DMPAPER_JENV_KAKU3_ROTATED
EnvMonarch	279 x 540	98.43 x 190.5	**3.875 x 7.5**	E	DMPAPER_ENV_MONARCH
EnvPersonal	261 x 468	92.08 x 165.1	**3.625 x 6.5**	E	DMPAPER_ENV_PERSONAL
EnvPRC1	289 x 468	**102 x 165**	4 x 6.5	E	DMPAPER_PENV_1
EnvPRC1Rotated	468 x 289	**165 x 102**	6.5 x 4	E, N	DMPAPER_PENV_1_ROTATED
EnvPRC2	289 x 499	**102 x 176**	4 x 6.9	E	DMPAPER_PENV_2
EnvPRC2Rotated	499 x 289	**176 x 102**	6.9 x 4	E, N	DMPAPER_PENV_2_ROTATED
EnvPRC3	354 x 499	**125 x 176**	4.9 x 6.9	E	DMPAPER_PENV_3
EnvPRC3Rotated	499 x 354	**176 x 125**	6.9 x 4.9	E, N	DMPAPER_PENV_3_ROTATED
EnvPRC4	312 x 590	**110 x 208**	4.33 x 8.2	E	DMPAPER_PENV_4
EnvPRC4Rotated	590 x 312	**208 x 110**	8.2 x 4.33	E, N	DMPAPER_PENV_4_ROTATED
EnvPRC5	312 x 624	**110 x 220**	4.33 x 8.66	E	DMPAPER_PENV_5
EnvPRC5Rotated	624 x 312	**220 x 110**	8.66 x 4.33	E, N	DMPAPER_PENV_5_ROTATED
EnvPRC6	340 x 652	**120 x 230**	4.7 x 9	E	DMPAPER_PENV_6
EnvPRC6Rotated	652 x 340	**230 x 120**	9 x 4.7	E, N	DMPAPER_PENV_6_ROTATED
EnvPRC7	454 x 652	**160 x 230**	6.3 x 9	E	DMPAPER_PENV_7
EnvPRC7Rotated	652 x 454	**230 x 160**	9 x 6.3	E, N	DMPAPER_PENV_7_ROTATED
EnvPRC8	340 x 876	**120 x 309**	4.7 x 12.2	E	DMPAPER_PENV_8
EnvPRC8Rotated	876 x 340	**309 x 120**	12.2 x 4.7	E, N	DMPAPER_PENV_8_ROTATED
EnvPRC9	649 x 918	**229 x 324**	9 x 12.75	E	DMPAPER_PENV_9
EnvPRC9Rotated	918 x 649	**324 x 229**	12.75 x 9	E, N	DMPAPER_PENV_9_ROTATED
EnvPRC10	918 x 1298	**324 x 458**	12.75 x 18	E	DMPAPER_PENV_10
EnvPRC10Rotated	1298 x 918	**458 x 324**	18 x 12.75	E, N	DMPAPER_PENV_10_ROTATED
EnvYou4	298 x 666	**105 x 235**	4.13 x 9.25	E	DMPAPER_JENV_YOU4
EnvYou4Rotated	666 x 298	**235 x 105**	9.25 x 4.13	E, N	DMPAPER_JENV_YOU4_ROTATED
Executive	522 x 756	184.2 x 266.7	**7.25 x 10.5**	N, V	DMPAPER_EXECUTIVE
FanFoldUS	1071 x 792	377.8 x 279.4	**14.875 x 11**		DMPAPER_FANFOLD_US
FanFoldGerman	612 x 864	215.9 x 304.8	**8.5 x 12**		DMPAPER_FANFOLD_STD_GERMAN
FanFoldGermanLegal	612 x 936	215.9 x 330	**8.5 x 13**		DMPAPER_FANFOLD_LGL_GERMAN

Table B.1 *MediaOptions Sorted By Name (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
Folio	595 x 935	**210 x 330**	8.27 x 13		DMPAPER_FOLIO
ISOB0	2835 x 4008	**1000 x 1414**	39.37 x 55.67	I	
ISOB1	2004 x 2835	**707 x 1000**	27.83 x 39.37	I	
ISOB2	1417 x 2004	**500 x 707**	19.68 x 27.83	I	
ISOB3	1001 x 1417	**353 x 500**	13.90 x 19.68	I	
ISOB4	709 x 1001	**250 x 353**	9.84 x 13.90	I	DMPAPER_ISO_B4
ISOB5	499 x 709	**176 x 250**	6.9 x 9.8	I	
ISOB5Extra	569.7 x 782	201 x 276	**7.9 x 10.8**	N, V	DMPAPER_B5_EXTRA
ISOB6	354 x 499	**125 x 176**	4.92 x 6.93	I	
ISOB7	249 x 354	**88 x 125**	3.46 x 4.92	I	
ISOB8	176 x 249	**62 x 88**	2.44 x 3.46	I	
ISOB9	125 x 176	**44 x 62**	1.73 x 2.44	I	
ISOB10	88 x 125	**31 x 44**	1.22 x 1.73	I	
Ledger	1224 x 792	431.8 x 279.4	**17 x 11**		DMPAPER_LEDGER
Legal	612 x 1008	215.9 x 355.6	**8.5 x 14**		DMPAPER_LEGAL
LegalExtra	684 x 1080	241.3 x 381	**9.5 x 15**	N, V	DMPAPER_LEGAL_EXTRA
Letter	612 x 792	215.9 x 279.4	**8.5 x 11**		DMPAPER_LETTER
Letter.Transverse	612 x 792	215.9 x 279.4	**8.5 x 11**		DMPAPER_LETTER_TRANSVERSE
LetterExtra	684 x 864	241.3 x 304.8	**9.5 x 12**	N, V	DMPAPER_LETTER_EXTRA
LetterExtra.Transverse	684 x 864	241.3 x 304.8	**9.5 x 12**		DMPAPER_LETTER_EXTRA_TRANSVERSE
LetterPlus	612 x 913.7	215.9 x 322.3	**8.5 x 12.69**		DMPAPER_LETTER_PLUS
LetterRotated	792 x 612	279.4 x 215.9	**11 x 8.5**	N	DMPAPER_LETTER_ROTATED
LetterSmall	612 x 792	215.9 x 279.4	**8.5 x 11**	S	DMPAPER_LETTERSMALL
MaxPage	*largest page*	*available on*	*this device*	N, V	
Monarch (use **EnvMonarch**)	279 x 540	98.43 x 190.5	**3.875 x 7.5**	E, N	DMPAPER_ENV_MONARCH
Note	612 x 792	215.9 x 279.4	**8.5 x 11**	S	DMPAPER_NOTE
Postcard	284 x 419	**100 x 148**	3.94 x 5.83		DMPAPER_JAPANESE_POSTCARD
PostcardRotated	419 x 284	**148 x 100**	5.83 x 3.94	N	DMPAPER_JAPANESE_POSTCARD_ROTATED
PRC16K	414 x 610	**146 x 215**	5.75 x 8.5		DMPAPER_P16K
PRC16KRotated	610 x 414	**215 x 146**	8.5 x 5.75	N	DMPAPER_P16K_ROTATED

Table B.1 *MediaOptions Sorted By Name (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
PRC32K	275 x 428	**97 x 151**	3.82 x 5.95		DMPAPER_P32K
PRC32KBig	275 x 428	**97 x 151**	3.82 x 5.95		DMPAPER_P32KBIG
PRC32KBigRotated	428 x 275	**151 x 97**	5.95 x 3.82	N	DMPAPER_P32KBIG_ROTATED
PRC32KRotated	428 x 275	**151 x 97**	5.95 x 3.82	N	DMPAPER_P32K_ROTATED
Quarto	**610 x 780**	215.9 x 275.1	8.5 x 10.83		DMPAPER_QUARTO
Statement	396 x 612	139.7 x 215.9	**5.5 x 8.5**		DMPAPER_STATEMENT
SuperA	643 x 1009	**227 x 356**	8.94 x 14		DMPAPER_A_PLUS
SuperB	864 x 1380	**305 x 487**	12 x 19.17		DMPAPER_B_PLUS
Tabloid	792 x 1224	279.4 x 431.8	**11 x 17**		DMPAPER_TABLOID
TabloidExtra	864 x 1296	304.8 x 457.2	**12 x 18**	V	DMPAPER_TABLOID_EXTRA

.

Table B.2 *MediaOptions Sorted By Size*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
A10	73 x 105	**26 x 37**	1.02 x 1.46	I, J	
ISOB10	88 x 125	**31 x 44**	1.22 x 1.73	I	
B10	91 x 127	**32 x 45**	1.26 x 1.77	J	
A9	105 x 148	**37 x 52**	1.46 x 2.05	I, J	
ISOB9	125 x 176	**44 x 62**	1.73 x 2.44	I	
B9	127 x 181	**45 x 64**	1.77 x 2.52	J	
A8	148 x 210	**52 x 74**	2.05 x 2.91	I, J	
ISOB8	176 x 249	**62 x 88**	2.44 x 3.46	I	
B8	181 x 258	**64 x 91**	2.52 x 3.58	J	
A7	210 x 297	**74 x 105**	2.91 x 4.13	I, J	
EnvC7	230 x 323	**81 x 114**	3.19 x 4.49	I, E	
ISOB7	249 x 354	**88 x 125**	3.46 x 4.92	I	
EnvChou4	255 x 581	**90 x 205**	3.54 x 8	E	DMPAPER_JENV_CHOU4
B7	258 x 363	**91 x 128**	3.58 x 5.04	J	
EnvPersonal	261 x 468	92.08 x 165.1	**3.625 x 6.5**	E	DMPAPER_ENV_PERSONAL
PRC32K	275 x 428	**97 x 151**	3.82 x 5.95		DMPAPER_P32K
PRC32KBig	275 x 428	**97 x 151**	3.82 x 5.95		DMPAPER_P32KBIG
Monarch (use **EnvMonarch**)	279 x 540	98.43 x 190.5	**3.875 x 7.5**	E	DMPAPER_ENV_MONARCH
Env9	279 x 639	98.4 x 225.4	**3.875 x 8.875**	E	DMPAPER_ENV_9
Postcard	284 x 419	**100 x 148**	3.94 x 5.83		DMPAPER_JAPANESE_POSTCARD
EnvPRC1	289 x 468	**102 x 165**	4 x 6.5	E	DMPAPER_PENV_1
EnvPRC2	289 x 499	**102 x 176**	4 x 6.9	E	DMPAPER_PENV_2
Comm10 (use **Env10**)	297 x 684	104.8 x 241.3	**4.125 x 9.5**	E, N	DMPAPER_ENV_10
A6	297 x 420	**105 x 148**	4.13 x 5.83	I, J	DMPAPER_A6
EnvYou4	298 x 666	**105 x 235**	4.13 x 9.25	E	DMPAPER_JENV_YOU4
EnvPRC4	312 x 590	**110 x 208**	4.33 x 8.2	E	DMPAPER_PENV_4
DL (use **EnvDL**)	312 x 624	**110 x 220**	4.33 x 8.66	I, E	DMPAPER_ENV_DL
EnvPRC5	312 x 624	**110 x 220**	4.33 x 8.66	E	DMPAPER_PENV_5
EnvItalian	312 x 652	**110 x 230**	4.33 x 9	E	DMPAPER_ENV_ITALY

Table B.2 *MediaOptions Sorted By Size (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
C6 (use **EnvC6**)	323 x 459	**114 x 162**	4.49 x 6.38	I, E	DMPAPER_ENV_C6
EnvC65	324 x 648	**114 x 229**	4.5 x 9	E	DMPAPER_ENV_C65
Env11	324 x 747	114.3 x 263.5	**4.5 x 10.375**	E	DMPAPER_ENV_11
EnvPRC6	340 x 652	**120 x 230**	4.7 x 9	E	DMPAPER_PENV_6
EnvChou3	340 x 666	**120 x 235**	4.72 x 9.25	E	DMPAPER_JENV_CHOU3
EnvPRC8	340 x 876	**120 x 309**	4.7 x 12.2	E	DMPAPER_PENV_8
Env12	342 x 792	120.7 x 279.4	**4.75 x 11**	E	DMPAPER_ENV_12
ISOB6	354 x 499	**125 x 176**	4.92 x 6.93	I	
EnvPRC3	354 x 499	**125 x 176**	4.9 x 6.9	E	DMPAPER_PENV_3
Env14	360 x 828	127 x 292.1	**5 x 11.5**	E	DMPAPER_ENV_14
B6	363 x 516	**128 x 182**	5.04 x 7.17	J	DMPAPER_B6_JIS
Statement	396 x 612	139.7 x 215.9	**5.5 x 8.5**		DMPAPER_STATEMENT
PRC16K	414 x 610	**146 x 215**	5.75 x 8.5		DMPAPER_P16K
PostcardRotated	419 x 284	**148 x 100**	5.83 x 3.94	N	DMPAPER_JAPANESE_POSTCARD_ROTATED
A6Rotated	420 x 297	**148 x 105**	5.83 x 4.13		DMPAPER_A6_ROTATED
DoublePostcardRotated	419.5 x 567	**148 x 200**	5.83 x 7.87	N	DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED
A5	420 x 595	**148 x 210**	5.83 x 8.27	I, J	DMPAPER_A5
A5.Transverse	420 x 595	**148 x 210**	5.83 x 8.27		DMPAPER_A5_TRANSVERSE
PRC32KBigRotated	428 x 275	**151 x 97**	5.95 x 3.82	N	DMPAPER_P32KBIG_ROTATED
PRC32KRotated	428 x 275	**151 x 97**	5.95 x 3.82	N	DMPAPER_P32K_ROTATED
EnvPRC7	454 x 652	**160 x 230**	6.3 x 9	E	DMPAPER_PENV_7
C5 (use **EnvC5**)	459 x 649	**162 x 229**	6.38 x 9.02	I, E	DMPAPER_ENV_C5
EnvPRC1Rotated	468 x 289	**165 x 102**	6.5 x 4	E, N	DMPAPER_PENV_1_ROTATED
A5Extra	492 x 668	**174 x 235**	6.85 x 9.25	N, V	DMPAPER_A5_EXTRA
EnvPRC2Rotated	499 x 289	**176 x 102**	6.9 x 4	E, N	DMPAPER_PENV_2_ROTATED
EnvISOB6	499 x 354	**176 x 125**	6.9 x 4.9	E	DMPAPER_ENV_B6
EnvPRC3Rotated	499 x 354	**176 x 125**	6.9 x 4.9	E, N	DMPAPER_PENV_3_ROTATED
EnvISOB5	499 x 709	**176 x 250**	6.9 x 9.8	E	DMPAPER_ENV_B5
ISOB5	499 x 709	**176 x 250**	6.9 x 9.8	I	
7x9	504 x 648	177.8 x 228.6	**7 x 9**		

Table B.2 *MediaOptions Sorted By Size (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
B6Rotated	516 x 363	**182 x 128**	7.17 x 5.04	N	DMPAPER_B6_JIS_ROTATED
B5	516 x 729	**182 x 257**	7.17 x 10.12	J	DMPAPER_B5
B5.Transverse	516 x 729	**182 x 257**	7.17 x 10.12		DMPAPER_B5_TRANSVERSE
Executive	522 x 756	184.2 x 266.7	**7.25 x 10.5**	N, V	DMPAPER_EXECUTIVE
DoublePostcard	567 x 419.5	**200 x 148**	7.87 x 5.83		DMPAPER_DBL_JAPANESE_POSTCARD
ISOB5Extra	569.7 x 782	201 x 276	**7.9 x 10.8**	N, V	DMPAPER_B5_EXTRA
8x10	576 x 720	203.2 x 254	**8 x 10**		
EnvChou4Rotated	581 x 255	**205 x 90**	8 x 3.54	E, N	DMPAPER_JENV_CHOU4_ROTATED
EnvPRC4Rotated	590 x 312	**208 x 110**	8.2 x 4.33	E, N	DMPAPER_PENV_4_ROTATED
A5Rotated	595 x 420	**210 x 148**	8.27 x 5.83	N	DMPAPER_A5_ROTATED
A4	595 x 842	**210 x 297**	8.27 x 11.69	I, J	DMPAPER_A4
A4.Transverse	595 x 842	**210 x 297**	8.27 x 11.69		DMPAPER_A4_TRANSVERSE
A4Small	595 x 842	**210 x 297**	8.27 x 11.69	S	DMPAPER_A4SMALL
Folio	595 x 935	**210 x 330**	8.27 x 13		DMPAPER_FOLIO
A4Plus	595 x 936	**210 x 330**	8.27 x 13		DMPAPER_A4_PLUS
PRC16KRotated	610 x 414	**215 x 146**	8.5 x 5.75	N	DMPAPER_P16K_ROTATED
Quarto	**610 x 780**	215.9 x 275.1	8.5 x 10.83		DMPAPER_QUARTO
EnvKaku3	612 x 785	**216 x 277**	8.5 x 10.9	E	DMPAPER_JENV_KAKU3
Letter	612 x 792	215.9 x 279.4	**8.5 x 11**		DMPAPER_LETTER
Letter.Transverse	612 x 792	215.9 x 279.4	**8.5 x 11**		DMPAPER_LETTER_TRANSVERSE
LetterSmall	612 x 792	215.9 x 279.4	**8.5 x 11**	S	DMPAPER_LETTERSMALL
Note	612 x 792	215.9 x 279.4	**8.5 x 11**	S	DMPAPER_NOTE
FanFoldGerman	612 x 864	215.9 x 304.8	**8.5 x 12**		DMPAPER_FANFOLD_STD_GERMAN
LetterPlus	612 x 913.7	215.9 x 322.3	**8.5 x 12.69**		DMPAPER_LETTER_PLUS
FanFoldGermanLegal	612 x 936	215.9 x 330	**8.5 x 13**		DMPAPER_FANFOLD_LGL_GERMAN
Legal	612 x 1008	215.9 x 355.6	**8.5 x 14**		DMPAPER_LEGAL
EnvPRC5Rotated	624 x 312	**220 x 110**	8.66 x 4.33	E, N	DMPAPER_PENV_5_ROTATED
EnvInvite	624 x 624	**220 x 220**	8.66 x 8.66	E	DMPAPER_ENV_INVITE
SuperA	643 x 1009	**227 x 356**	8.94 x 14		DMPAPER_A_PLUS
9x11	648 x 792	228.6 x 279.4	**9 x 11**		DMPAPER_9X11

Table B.2 *MediaOptions Sorted By Size (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
9x12	648 x 864	228.6 x 304.8	**9 x 12**		
ARCHA	648 x 864	228.6 x 304.8	**9 x 12**		
C4 (use **EnvC4**)	649 x 918	**229 x 324**	9.02 x 12.75	I, E	DMPAPER_ENV_C4
EnvPRC9	649 x 918	**229 x 324**	9 x 12.75	E	DMPAPER_PENV_9
EnvPRC6Rotated	652 x 340	**230 x 120**	9 x 4.7	E, N	DMPAPER_PENV_6_ROTATED
EnvPRC7Rotated	652 x 454	**230 x 160**	9 x 6.3	E ,N	DMPAPER_PENV_7_ROTATED
EnvYou4Rotated	666 x 298	**235 x 105**	9.25 x 4.13	E, N	DMPAPER_JENV_YOU4_ROTATED
EnvChou3Rotated	666 x 340	**235 x 120**	9.25 x 4.72	E, N	DMPAPER_JENV_CHOU3_ROTATED
A4Extra	667 x 914	235.5 x 322.3	**9.27 x 12.69**	N, V	DMPAPER_A4_EXTRA
EnvKaku2	680 x 941	**240 x 332**	9.45 x 13	E	DMPAPER_JENV_KAKU2
LetterExtra	684 x 864	241.3 x 304.8	**9.5 x 12**	N, V	DMPAPER_LETTER_EXTRA
LetterExtra.Transverse	684 x 864	241.3 x 304.8	**9.5 x 12**	N, V	DMPAPER_LETTER_EXTRA_TRANSVERSE
LegalExtra	684 x 1080	241.3 x 381	**9.5 x 15**	N, V	DMPAPER_LEGAL_EXTRA
EnvISOB4	708 x 1001	**250 x 353**	9.84 x 13.9	E	DMPAPER_ENV_B4
ISOB4	709 x 1001	**250 x 353**	9.84 x 13.90	I	DMPAPER_ISO_B4
10x11	720 x 792	254 x 279.4	**10 x 11**		DMPAPER_10X11
10x13	720 x 936	254 x 330.2	**10 x 13**		
10x14	720 x 1008	254 x 355.6	**10 x 14**		DMPAPER_10X14
B5Rotated	729 x 516	**257 x 182**	10.12 x 7.17	N	DMPAPER_B5_JIS_ROTATED
B4	729 x 1032	**257 x 364**	10.12 x 14.33	J	DMPAPER_B4
EnvKaku3Rotated	785 x 612	**277 x 216**	10.9 x 8.5	E, N	DMPAPER_JENV_KAKU3_ROTATED
LetterRotated	792 x 612	279.4 x 215.9	**11 x 8.5**	N	DMPAPER_LETTER_ROTATED
Tabloid	792 x 1224	279.4 x 431.8	**11 x 17**		DMPAPER_TABLOID
A4Rotated	842 x 595	**297 x 210**	11.69 x 8.27	N	DMPAPER_A4_ROTATED
A3	842 x 1191	**297 x 420**	11.69 x 16.54	I, J	DMPAPER_A3
A3.Transverse	842 x 1191	**297 x 420**	11.69 x 16.54		DMPAPER_A3_TRANSVERSE
12x11	864 x 792	304.8 x 279.4	**12 x 11**		DMPAPER_12X11
ARCHB	864 x 1296	304.8 x 457.2	**12 x 18**		
TabloidExtra	864 x 1296	304.8 x 457.2	**12 x 18**	V	DMPAPER_TABLOID_EXTRA
SuperB	864 x 1380	**305 x 487**	12 x 19.17		DMPAPER_B_PLUS

Table B.2 *MediaOptions Sorted By Size (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
EnvPRC8Rotated	876 x 340	**309 x 120**	12.2 x 4.7	E, N	DMPAPER_PENV_8_ROTATED
A3Extra	913 x 1262	**322 x 445**	12.67 x 17.52	N, V	DMPAPER_A3_EXTRA
A3Extra.Transverse	913 x 1262	**322 x 445**	12.67 x 17.52	N, V	DMPAPER_A3_EXTRA_TRANSVERSE
EnvPRC9Rotated	918 x 649	**324 x 229**	12.75 x 9	E, N	DMPAPER_PENV_9_ROTATED
EnvC3	918 x 1296	**324 x 458**	12.75 x 18.03	I, E	DMPAPER_ENV_C3
EnvPRC10	918 x 1298	**324 x 458**	12.75 x 18	E	DMPAPER_PENV_10
EnvKaku2Rotated	941 x 680	**332 x 240**	13 x 9.45	E, N	DMPAPER_JENV_KAKU2_ROTATED
ISOB3	1001 x 1417	**353 x 500**	13.90 x 19.68	I	
B4Rotated	1032 x 729	**364 x 257**	14.33 x 10.12	N	DMPAPER_B4_JIS_ROTATED
B3	1032 x 1460	**364 x 515**	14.33 x 20.28	J	
FanFoldUS	1071 x 792	377.83x 279.4	**14.875 x 11**		DMPAPER_FANFOLD_US
15x11	1080 x 792	381 x 279.4	**15 x 11**		DMPAPER_15X11
A3Rotated	1191 x 842	**420 x 297**	16.54 x 11.69	N	DMPAPER_A3_ROTATED
A2	1191 x 1684	**420 x 594**	16.54 x 23.39	I, J	DMPAPER_A2
Ledger	1224 x 792	431.8 x 279.4	**17 x 11**		DMPAPER_LEDGER
AnsiC	1224 x 1584	431.8 x 558.8	**17 x 22**		
ARCHC	1296 x 1728	457.2 x 609.6	**18 x 24**		DMPAPER_CSHEET
EnvPRC10Rotated	1298 x 918	**458 x 324**	18 x 12.75	E, N	DMPAPER_PENV_10_ROTATED
EnvC2	1298 x 1837	**458 x 648**	18.03 x 25.51	I , E	
ISOB2	1417 x 2004	**500 x 707**	19.68 x 27.83	I	
B2	1460 x 2064	**515 x 728**	20.28 x 28.66	J	
AnsiD	1584 x 2448	558.8 x 863.6	**22 x 34**		
A1	1684 x 2384	**594 x 841**	23.39 x 33.11	I, J	
ARCHD	1728 x 2592	609.6 x 914.4	**24 x 36**		DMPAPER_DSHEET
EnvC1	1837 x 2599	**648 x 917**	25.51 x 36.10	I, E	
ISOB1	2004 x 2835	**707 x 1000**	27.83 x 39.37	I	
B1	2064 x 2920	**728 x 1030**	28.66 x 40.55	J	
A0	2384 x 3370	**841 x 1189**	33.11 x 46.81	I, J	
AnsiE	2448 x 3168	863.6 x 1118	**34 x 44**		

Table B.2 *MediaOptions Sorted By Size (Continued)*

mediaOption	Size (pts)	Size (mm)	Size (inches)	Notes	#define
ARCHE	2592 x 3456	914.4 x 1219	**36 x 48**		DMPAPER_ESHEET
EnvC0	2599 x 3676	**917 x 1297**	36.10 x 51.06	I, E	
ISOB0	2835 x 4008	**1000 x 1414**	39.37 x 55.67	I	
B0	2920 x 4127	**1030 x 1456**	40.55 x 57.32	J	
MaxPage	*largest page*	*size available*	*on this device*	N, V	

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 (character set 0 or Western), and MacStandard (Script Manager script 0). 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 C.1 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 C.2, C.3, and C.4 contain only the differences between the three encoding vectors, and could be the basis for conversion tables in a document manager. Table C.2 is indexed by the character code and name of each character in the WindowsANSI encoding vector. Table C.3 is indexed by the character code and name of each character in the MacStandard encoding vector. Table C.4 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 C.1 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 C.1 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
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 C.1 All Encodings Indexed By Byte Code (Continued)

Code	*WindowsANSI*	*ISOLatin1*	*MacStandard*
159	Ydieresis	caron	udieresis
160	space[a]	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[b]	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
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

a. Some PostScript fonts may include code 160 as a non-breaking space named nbspace.

b. Some PostScript fonts may include code 173 as a soft hyphen named sfthyphen.

C.2 Conversions from WindowsANSI Encoding

In Table C.2, 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 C.2 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*
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*
guillemotleft	**171**	199	*same*
logicalnot	**172**	194	*same*

Table C.2 Conversions from WindowsANSI Encoding (Continued)

Character Name	***ANSI***	*Mac*	*ISOLatin1*
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*
Ograve	**210**	241	*same*
Oacute	**211**	238	*same*

Table C.2 Conversions from WindowsANSI Encoding (Continued)

Character Name	***ANSI***	*Mac*	*ISOLatin1*
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 C.3, 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 C.3 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
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
Agrave	**203**	192	192
Atilde	**204**	195	195

Table C.3 Conversions from MacStandard Encoding

Character Name	***Mac***	*ANSI*	*ISOLatin1*
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
macron	**248**	175	149
breve	**249**	*n/a*	150
dotaccent	**250**	*n/a*	151

Table C.3 Conversions from MacStandard Encoding

Character Name	***Mac***	*ANSI*	*ISOLatin1*
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 C.4, 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 C.4 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
cedilla	**184**	*same*	252

Table C.4 Conversions from ISOLatin1 Encoding

Character Name	***ISOLatin1***	*ANSI*	*Mac*
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
Ucircumflex	**219**	*same*	243
Udieresis	**220**	*same*	134

Table C.4 Conversions from ISOLatin1 Encoding

Character Name	***ISOLatin1***	*ANSI*	*Mac*
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: Manufacturer's Prefix List and *Manufacturer Strings

The first column of Table D.1 contains the formal, legal name of device manufacturers who support Adobe PostScript in their devices.

The second column of Table D.1 contains a list of two-letter prefixes that have been assigned to device manufacturers. The manufacturer's assigned prefix composes the first two characters of an initial PPD file name, the first two characters of *PCFileName, and the first two characters after the asterisk of any main keywords created by the manufacturer. Each manufacturer must have a unique prefix and the prefix must be the same in all PPD files for devices from that manufacturer. For example, all PPD files built for Agfa devices will have filenames and *PCFileName values that start with the character sequence "AG", and any main keywords created by Agfa will start with the character sequence *AG, as in *AGHalftone. See *PCFileName in section 5.3 for advice on naming PPD files. See section 5.1 for information about creating and properly prefixing keywords.

The third column of Table D.1 contains the value of *Manufacturer (see section 5.3), if known. The fourth column of Table D.1 contains the value of the tag *icHeader.manufacturer* in ICC color characterization profiles belonging to this manufacturer in the Windows environment, if known.

Note *Because the *Manufacturer keyword is new in the 4.3 version of this specification, very few *Manufacturer strings and icHeader.manufacturer profile tags are known at this time. By the time the next version of this specification is released, this table will be much more complete.*

Note *To builders of PPD files: If you do not have an assigned *Manufacturer string, before choosing one, please consult Table D.1 to avoid name conflicts. Also, if you plan to install ICC color profiles in the Windows environment, certain restrictions are placed on the first four characters of these names and they must not conflict with the first four characters of other manufacturer's names. Please read the notes for *Manufacturer, *ModelName, and *ShortNickName in section 5.3.*

For updates to this list, please contact the appropriate address on the front cover of this document.

Table D.1 *Assigned prefixes and *Manufacturer strings as of February 9, 1996*

Company Name	*Prefix*	*Manufacturer	*icHeader.manufacturer*
3M Corporation	3M		
Adobe Systems Inc.	AD	Adobe	ADOB
Agfa-Gevaert N.V. (includes Agfa-Matrix, Agfa-Compugraphic, and Miles Inc.)	AG		
Apple Computer, Inc.	AP	Apple	APPL
AST Research Inc.	AS	AST	AST
Autologic Incorporated (a subsidiary of Volt Information Sciences)	AU		
Barco Graphics	BC		
Birmy Graphics Corporation	BG		
Bull HN Information Systems Italia S.P.A.	BU	Bull	BULL
Cactus	CC		
CalComp, Inc.	CA		
Canon, Inc.	CN	Canon	CANO
Colorbus Software	CB		
Colossal Graphics Inc.	CG		
Compaq Computer Corporation	CP	Compaq	COMP
Crosfield Electronics Limited	CF		
Dainippon Screen Mfg. Co. Ltd.	DS		
Dataproducts Corporation	DP	Dataproducts	DATA
Digital Equipment Corporation	DC		
DuPont *see* E.I.DuPont	—	—	—
Eastman Kodak Company and Diconix (a Division of Kodak)	KD	Kodak	KODA
Eicon Technology Corporation (includes Escher-Grad Inc.)	EG		
E.I. DuPont deNemours and Company	DU		
EFI, Inc. (Electronics For Imaging, Inc.)	EF		
Epson *see* Seiko Epson	—	—	—
Fargo Electronics, Inc.	FE		

Table D.1 *Assigned prefixes and *Manufacturer strings as of February 9, 1996 (Continued)*

Company Name	*Prefix*	*Manufacturer	*icHeader.manufacturer*
Fujitsu, Inc.	FU		
Fuji Film	FF		
Fuji Xerox	FX		
GCC Technologies	GC		
Gestetner Lasers Pty. Limited	GS		
Hewlett Packard Company	HP	HP	HP
Hitachi Koki Co., Ltd.	HK		
IDT	ID		
Indigo	IN		
Integrated Computer Solutions, Inc.	IC		
International Business Machines Corp.	IB	IBM	IBM
Kodak *see* Eastman Kodak	—	—	—
Lasergraphics	LG		
Lexmark International	LX	Lexmark	LEXM
Lincoln, A.J.	LI		
Linotype-Hell AG	LH		
Management Graphics	MG		
Mannesmann Scangraphic GmbH	SC	Mannesmann	MANN
Matsushita Electric Industrial Co., Ltd.	MT		
Mitsubishi Electric Corporation	ME		
Monotype Corporation PLC	MO		
NEC Corporation (includes NEC Information Systems, Inc. and NEC Technologies, Inc.)	NC	NEC	NEC
Network Computing Devices, Inc.	ND		
Newgen Corporation	NW		
NeXT Computer, Inc.	NX		
Nihon System Gijutsu Co.	NP		
Oce Graphics USA Inc. (formerly Schlumberger)	OC	Oce	OCE
Ohio Electronic Engravers	OE		

Table D.1 *Assigned prefixes and *Manufacturer strings as of February 9, 1996 (Continued)*

Company Name	*Prefix*	*Manufacturer	*icHeader.manufacturer*
Oki Electric Industry Co. (includes Okidata)	OK	Oki	OKI
Optronics, a Division of Intergraph	OP		
Panasonic USA	PA	Panasonic	PANA
PIX Computer Systems GmbH	PX		
PrePRESS Solutions, Inc.	PP		
QMS, Inc.	QM	QMS	QMS
Radius	RA		
Ricoh Company, Ltd.	RI	Ricoh	RICO
Scangraphic *see* Mannesmann	—	—	—
Scitex Corporation Ltd.	SX		
Seiko Epson Corporation	EP	Epson	EPSO
Seiko Instruments USA, Inc.	SK	Seiko	SEIK
Silicon Graphics	SG		
Sony Corporation	SO		
Sun Microsystems, Inc.	SN		
SuperMac Technology, Inc.	SM		
Tektronix	TK	Tektronix	TEKT
Texas Instruments Inc.	TI	TI	TI
Total Integration, Inc.	TO		
Unisys	UN		
Ultimate Technographics	UT		
Varityper Inc. (*now* PrePRESS Solutions, Inc.)	VT		
VerTec Solutions, Inc.	VS		
Visual Edge Software Ltd.	VE		
Wang Laboratories Inc.	WA	Wang	WANG
Xante Corporation	XT		
Xerox Corporation	XR	Xerox	XERO

Appendix E: Changes Since Earlier Versions

E.1 Changes since Version 4.2, March 29, 1994

New Material

- Added the following new *required* keywords:

  *Manufacturer

- Added the following new non-required keywords:

*1284DeviceID	*1284Modes	*CloseSubGroup
*ContoneOnly	*DefaultExitJamRecovery	*DefaultHalftoneType
*DefaultLeadingEdge	*DefaultUseHWMargins	*ExitJamRecovery
*?ExitJamRecovery	*FCacheSize	*FDirSize
*HalftoneName	*?InstalledMemory	*LeadingEdge
*NonUIOrderDependency	*NonUIConstraints	*OpenSubGroup
*PageDeviceName	*QueryOrderDependency	*ReferencePunch
*RenderingIntent	*SuggestedManualFeedTimeout	
*UseHWMargins		

- Added new *section 4.5, "Summary of Rules for *Default Keywords"*, to get all the information about *Default keywords into one place.

- Added new *section 5.1, "Creating Your Own Keywords",* to document how device manufacturers can create their own main and option keywords when building PPD files for their devices.

- Added new *section 6.3, "Examples of Custom Page Size Code*", to assist builders of PPD files with writing *CustomPageSize invocation code for both roll-fed and cut-sheet devices.

- Added new *section 7, "PPD File Summary"*, for PPD file builders, containing a summary of what goes into a PPD file, including changes to make for a Kanji PPD file.

- Added new *Appendix D: Manufacturer's Prefix List and *Manufacturer Strings*, to list PPD filename & keyword prefixes assigned to device manufacturers, known string values for *Manufacturer, and icHeader.manufacturer color profile tags, if known.

- Renamed old Appendix D (this appendix) to Appendix E. Removed *Changes since February 14, 1992*, as it only fixed minor typographical errors and provided no useful information.

- Combined Appendix B with Appendix A, renamed *Appendix A: Keyword Categories*. Added new *Appendix B: Registered mediaOption Keywords*. Moved all lists of *mediaOption* option keywords (page sizes) from section 5.9, *Media Option Keywords*, into new Appendix B, and combined lists into two tables, sorted by name and size.

- New *Appendix B*: Introduced the prefix Env to designate envelope page sizes. The following sizes had the prefix Env added to reflect this change:C0, C1, C2, C3, C4, C5, C6, C7, DL, and Monarch. Also, Comm10 was changed to Env10. These names were **not** changed in existing PPD files. The old names are still valid, but builders of new PPD files are encouraged to use the new names for easy recognition of envelopes by print managers.

- Indexed all previously deleted keywords for reference to old PPD files.

Changes to Existing Material

- Made *ShortNickName a *required* keyword. It was not required in previous versions of this specification. Added it to various examples. Also clarified that the *Default versions of required keywords are also required. For example, *DefaultPageSize is required.

- Throughout document, changed all examples using **setpagedevice** to use recommended method of construction dictionaries, by using **(<<) cvx exec** etc. Updated many examples from Level 1 to Level 2 code. Also added audience designators to many Notes: *To builders of PPD files*, and *To application developers*. Consolidated descriptions of main keywords with their associated *Default and query keywords, where it made sense to do so. Added Unknown to all *Default values. Specified valid return values, including newline, for all query keywords in a standard format.

- Moved *ASCII Code Chart* and *Definition of Terms* closer to front of document. Many other sections were moved, reorganized, and renumbered to be more internally consistent and consistent with the actual structure of most PPD files.

- *Error Handling:* Removed comments on file portability; covered elsewhere.

- *Order Dependencies*: Edited to clarify that the code ordering guidelines are only needed if a parser does not use the *OrderDependency statements provided in the PPD file. Removed redundant examples.

- *Local Customization Files*: Clarified that local customization files must include the minimal set of required keywords, and that any customization of UI keywords must include the entire *OpenUI/*CloseUI entry. Added warnings about creation & use.

- *Definition of Terms*: Moved closer to front of document. Removed comments about keyword registration, since this is now covered in section 5.1. Added note to state that query keywords do not exist for every main keyword. Added definitions for, *Default keywords, and *stand-alone* *Default keywords. Changed allowable byte-code ranges to allow 8-bit ASCII in translation strings and QuotedValues. Changed definition of *in-range* and *out-of-range* byte codes.

- *General Parsing Summary:* Added material about 8-bit ASCII byte codes.

- *Main Keywords*: Changed 3rd paragraph to make it clear that *Default keywords can appear alone in a PPD file, without a corresponding invocation or query keyword. Moved *required keywords list* from *Parsing Summary* to beginning of section 5.

- *Option Keywords*: Slightly reworded beginning of section to reflect changes in how option keywords are created.

- *Syntax of Values*, under *Parsing Summary for Values*: Changed several bullets to clarify use of stand-alone *Default keywords and use of Unknown. Added note to warn parsers about 8-bit ASCII appearing in certain QuotedValues in translated PPD files. Throughout document, changed the use of "StringValues" to "StringValue with multiple components separated by white space" where appropriate.

- *Translation String Syntax*: Clarified that *Default keywords may have a translation string on their value only if they are stand-alone keywords. Clarified that a given option keyword should have the same translation string across closely related entries such as *PageSize, *PageRegion, *ImageableArea, and *PaperDimension. Added note to warn parsers about 8-bit ASCII appearing in certain QuotedValues in translated PPD files. Added description of when hex must be used and when 8-bit ASCII may be used.

- *PostScript Language Sequences*: Added recommendations on writing Level 2 code that reduces the risk of errors if sent to a Level 1 device. Added advice on efficient Level 2 dictionary construction. Removed *?PageRegion query in example, as this query keyword does not exist. Clarified example of how *End is used.

- *PPD File Structure*: Added *Manufacturer and *ShortNickName to list of required keywords usually found near front of file. Added info about PPD file size limits and keyword content imposed by common print managers.

- *Syntax of Specification, General Syntax*: Added that PostScript operators and dictionary keys appear in boldface type. In *Elementary Types*, under *query*, added that valid return values are defined for each query, that translation strings are not allowed on return values, and that return values must terminate with a newline. Added new type *text* for strings with spaces. In *Standard Option Values for Main Keywords*, under Unknown, added that stand-alone *Default keywords can't have value of Unknown.

- *General Information Keywords*: Added more examples to most keywords; added guidelines for PPD file builders to many keywords. Changed the first part of a *PCFileName from an upper limit of 8 characters to specify that it must be 8 characters; added Adobe's file naming conventions and examples. Added options of ISOLatin2 and ISOLatin5 to *LanguageEncoding and Turkish to *LanguageVersion. Added explanation about ICC color profile matching to *ModelName and *ShortNickName. Changed value type of *ShortNickName, *ModelName, *NickName from *string* to *text* to accommodate spaces in value.

- *Emulators and Protocols:* From *Emulators, removed statement "Multiple *Emulators statements may appear." There is no need for multiple *Emulators statements, since *Emulators can list multiple emulators in a single StringValue. Explained odd syntax of *StopEmulator_ and *StartEmulator_ keywords.

- *Structure Keywords, *UIConstraints*: Added new rule that *UIConstraints can no longer be used with non-UI keywords; instead, the new keyword *NonUIConstraints should be used. Also, *UIConstraints should be reciprocated and should be considered reciprocated by print managers even if the reciprocal constraint is missing.

- *Structure Keywords, *OrderDependency:* *OrderDependency may only be used with UI keywords and the new keyword *NonUIOrderDependency must be used with non-UI keywords. AnySetup should be used for the *section* value if a specific section is not required. Adobe strongly recommends using *OrderDependency in every *OpenUI and *JCLOpenUI entry. Changed to say that *order* numbers define the order *within a section*, rather than across all sections, to accommodate printing the 1st page from a different bin, when the *InputSlot code is not emitted until the 2nd page. Added note about using the same *order* number if possible & how print managers can use this to reduce the number of **setpagedevice** calls and improve performance. Removed this statement regarding code fragments that don't have *OrderDependency statements: "[Such fragments] 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.”. This left only the statement that non-ordered fragments should be emitted after ordered fragments.

- *Structure Keywords*: *OpenGroup/*CloseGroup may not be nested; the new keywords *OpenSubGroup/*CloseSubGroup should be used for nesting groups within groups. Removed requirement that main keywords surround by *OpenUI/*CloseUI be documented in the PPD spec, as OEMs can invent their own main keywords. Added reference to *JCLOpenUI/*JCLCloseUI. In *Resolution example at end of *OrderDependency description, changed 2504 to 2504dpi. Rewrote entire section to be less verbose.

- *Symbolic References to Data*: Removed unnecessary titles on examples in description of *SymbolValue.

- *Media Option Keywords*: Removed lists of *mediaOptions* to Appendix B. Added explanation of how Transverse qualifier should and should not be used. Added explanations for other common qualifiers and page size substrings.

- *Custom Page Sizes*: Made substantial changes to how custom page sizes are handled on cut-sheet devices. Orientation, WidthOffset, and HeightOffset are now allowed. All parameters are now defined the same for roll-fed & cut-sheet media. Orientation can now be used with *HWMargins to figure out which edge is leading into the device, so the imageable area can now be more accurately determined by subtracting the correct values of *HWMargins from the correct edges of the page. Added section for print manager authors on how to handle custom page sizes. All complete examples of custom page size entries were moved to section 6.4. The keyword *VariablePaperSize was removed from the specification, as it was superceded by *CustomPageSize in version 4.0.

- *Media Handling Features, *InputSlot*: Added Microsoft #defines for input slots to list of *inputSlotOptions*. Removed LargeFormat as an *inputSlotOption* because it duplicated the function of AnyLargeFormat and was not used in any existing PPD files. Added note to *InputSlot about combining functionality such as *MediaType and *ManualFeed with *InputSlot. Added example of *InputSlot entry for single-slot or roll-fed devices.

- *Media Handling Features*: Removed OnlyOne as an option for *OutputBin, as this keyword should be omitted if there is only one output bin. Rewrote *RequiresPageRegion to clarify that *PageRegion code may be required for reasons other than input slots not sensing page size. Added note to *DefaultOutputBin; it should not be present without *OutputBin. To *DefaultOutputOrder, added that if this keyword is stand-alone, its value may not be Unknown.

- *Finishing Features*: Added option EndOfPage to *Slipsheet. Removed UI symbol from *InsertSheet, as this keyword requires the print manager to build a special UI to accommodate it, so it cannot be blindly parsed and displayed. Noted symbol removal. Added examples to folding, binding, and stapling keywords of how to write *UIConstraints so a print manager would understand the interdependencies between them.

- *Resolution and Appearance Control*: Combined descriptions of *ScreenAngle, *ScreenFreq, and *DefaultScreenProc. Added material to these keywords related to *DefaultHalftoneType and *ContoneOnly. Added note to description of *DefaultResolution to clarify that this keyword may appear alone, without *Resolution or *SetResolution. Changed *ResScreenFreq and *ResScreenAngle to say that they should be omitted in the PPD file of a Level 2 device if the default Halftone dictionary type is not "1". In *BitsPerPixel, deleted requirement that one of the options be None. Changed definition of None from "1 bit per pixel" to "lowest number of bits per pixel". Added options Off, On, True, False.

- *Color Issues*: Under *ColorRenderDict, changed how color rendering dictionaries are named, to correspond to the new Adobe CRD naming convention. Clarified how a CRD can be added to the device via *ColorRenderDict.

- *Font Related Keywords*: Clarified that fonts in the *Font list do not have to be Type 1 fonts. Added many new *charset* and *encoding* options to cover composite and CID-keyed fonts. Clarified how to deal with aftermarket plug-in fonts and host-downloaded fonts. Removed recommendation to put these in local customization file. Added note about future use of *Font with *NonUIConstraints. Added note to *?FontList and *?FontQuery about slowing down query responses.

- *Printer Messages*: Under *Source, added that this keyword lists the names of the communications channels. Changed the value type of *PrinterError, *Status, and *Message from *string* to *text*, to accommodate spaces in the value.

- *Features Accessible Only Through Job Control Language*: Added advice against including both the PS and JCL methods of invoking a given feature, if both methods exist. Added Unknown to list of valid return values from queries.

- *Sample PPD File Structure*: Added examples of custom page size entries and advice on how to write them. Removed Level 1 examples. Expanded Level 2 color printer and imagesetter examples.

- *Appendix A.2 (formerly Appendix B)*: Removed the following keywords from the list of optionless repeated keywords (these are optionless, but not repeated): *Emulators, *Extensions, *FaxSupport, and *Protocols.

- *Appendix C, Table C.2*: Added “character set 0 or Western” to definition of WindowsANSI and “Script Manager script 0” to definition of MacStandard. Removed byte codes 157 and 158 from the WindowsANSI encoding table, because these byte codes are unused, not bullets as was previously stated. Added footnotes about codes 160 and 173 to the WindowsANSI table (these codes may have different names than shown in the table).

- Globally changed several values that were recorded as “invocation code” to simply “invocation”. Fixed various formatting errors and typos and made minor rewording changes. Improved indexing.

- Changed spec version number from 4.2 to 4.3.

E.2 Changes since Version 4.1, April 9, 1993

- Changed spec version number from 4.1 to 4.2 in all appropriate places.

- Added the following new keywords:

*PrintPSErrors	*SuggestedJobTimeout	*SuggestedWaitTimeout
*ResScreenAngle	*ResScreenFreq	*InstalledMemory
*DefaultInstalledMemory		

- Added a new option TrueImage to *TTRasterizer.

- Moved description of *VMOption from section 5.4 to section 5.22, *System Management*, after the description of *FreeVM. Removed examples of using *UIConstraints on *VMOption to show how much VM is available (this method has been replaced by *InstalledMemory). Rewrote description of *FreeVM for clarity & accuracy with regard to *VMOption.

- Made the following changes to section 5.7, *Installable Options*: It is now legal, when necessary, to have named keywords, like *InstalledMemory, in the InstallableOptions group (instead of the generic *Option1 type of keyword). It is now legal to have PostScript code in the value of an entry in the Installable-Options group. If there is such code, the entry must also have an *OrderDepen-dency statement. Removed section on Keyword-Value pairs; they are no longer recommended in *UIConstraints.

- Added paragraph to description of *UIConstraints to say that constraints should only be used with UI keywords.

- Section 2.1, last paragraph: Removed the following sentence from the end of the description of defaults, as the PPD specification should not be dic-tating print manager behavior, only recommending it: “Print managers should ensure that if the user selects nothing else, the defaults shown in the user interface are invoked.” Replaced with description of how some print managers behave regarding defaults. Removed similar statement dictating print manager behavior from last paragraph of section 2.6.

- Section 5.9: Under Folio page size, changed incorrect reference to 8.5”x13” page to correct metric size of 210mm x 330mm. Changed point size of page from [595 936] to [595 935] for greater accuracy. Changed imageable area description to “approximate”. Removed references to “folio sheet” and “quarto sheet” (under the definition of Quarto page size).

- Section 5.14: Fixed typo “F*” at beginning of several keywords.

- *Appendix A*: Removed *AccurateScreensSupport from the list, as it was never an *OpenUI keyword. Added *InstalledMemory.

E.3 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 introduction 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.

E.4 Changes since Version 3.0, dated March 8, 1989

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:* 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. 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* introduction, 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 (shown here) throughout document to mark keywords that should be bracketed with *OpenUI/*CloseUI.

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

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 was 300dpi. This change is necessary to accommodate printers with anamorphic resolution.
- *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.
- *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.

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. Moved information on device resolution to new section *Resolution and Appearance Control*.

- *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. Explained 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.

- *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. 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*: Error was added as a legal value for *DefaultFont. Added charset and status fields to *Font, with explanatory table for status.

- *Printer Messages*: General rewording throughout section. Under *Message, added "Messages that appear under *Status or *PrinterError should not be repeated here." Clarified that a message may appear under both *Status and *PrinterError, added examples of entries and translation strings to *PrinterError, 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. The description of *Password was changed to refer to the *current* password instead of the *default* password. *DeviceAdjustMatrix should be commented out if it is not used. Added reference to *Localization* section to *DeviceAdjustMatrix).

- Cleaned up all sample code to eliminate "begin...end" so no dictionaries are left on the stack if the code fails.

Index

D

J, K

L

M

S

T