PostScript® Language Reference Manual Supplement

Adobe Systems Printer Developer Guide

For Version 2017

1 April 1996

Adobe Systems Incorporated

Corporate Headquarters
1585 Charleston Road
P.O. Box 7900
Mountain View, CA 94039-7900
(415) 961-4400 Main Number

Eastern Regional Office
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120

Adobe Systems Europe Limited
Adobe House, Mid New Cultins
Edinburgh EH11 4DU
Scotland, United Kingdom
+44-131-453-2211

Adobe Systems Japan
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150 Japan
+81-3-5423-8100

PN LPS5217

Contents

List of Figures

List of Tables

CHAPTER 1

Overview

1.1 Introduction

The main purpose of the *PostScript® Language Reference Manual Supplement* is to provide a supplement to the *PostScript Language Reference Manual, Second Edition* describing PostScript language Level 2 changes since the publication of the manual. Specifically, it describes:

- Additions to the standard page device, user, system and device parameters (chapters 2 and 3).
- New resource categories and instances (chapter 4).
- Other extensions to PostScript language Level 2 such as the Type 6 halftone and Type 42 font dictionaries (chapter 5).

In addition, this document lists the Level 1 compatibility operators and procedures that Adobe® supports in all PostScript language Level 2 implementations (chapter 6).

The intended audience for this supplement consists of independent software vendors (ISVs) who want to write PostScript language device drivers that can be used for more than one type of device. This document will help ISVs produce drivers that support all of the features and capabilities of existing and future PostScript output devices.

1.1.1 PostScript Language Level 2 Parameters

Level 2 of the PostScript language introduces several operators that take dictionaries as arguments and return dictionaries as results. The key-value pairs in these dictionaries are referred to as parameters because their values typically select optional features or control the operation of some part of the PostScript language implementation. The use of dictionaries as containers for parameters provides an extensible method of adding support for new features by adding a new parameter key to the appropriate dictionary. This approach avoids adding new operators to the language on a per-feature basis, thereby maintaining the device-independence of the PostScript language.

Specific PostScript language implementations include only the parameters that pertain to that product. It is not intended that all of the parameters described in this supplement be present in all products. Once a parameter is defined in any product, it is always used for the same feature in any subsequent products that support it.

Four classes of parameters exist in PostScript language Level 2: page device, user, system and device parameters. Each class corresponds to a pair of PostScript operators: one that returns the current values of a set of parameters and one that takes as an argument a collection of parameters that are to be set. These operators are:

currentpagedevice	**setpagedevice**
currentuserparams	**setuserparams**
currentsystemparams	**setsystemparams**
currentdevparams	**setdevparams**

In terms of functionality, parameters fall into two broad categories. The first category corresponds to printing capabilities (for example, optional trays, duplex and collating, among others). These are the *page device parameters.* The second category corresponds to the operation and behavior of the PostScript interpreter. These are the *interpreter parameters*, which include the system, user and device parameters. Chapter 2 describes the new page device parameters and chapter 3 describes the new interpreter parameters that have been added since the publication of *PostScript Language Reference Manual, Second Edition*.

1.1.2 PostScript Language Level 2 Resources

In Level 2, PostScript language objects, such as fonts, patterns, filters and so on, can be managed as open-ended collections of resources. Chapter 4 lists the resources that have been added since the publication of the *PostScript Language Reference Manual, Second Edition*.

1.1.3 Other Extensions to PostScript Language Level 2

The PostScript language continues to evolve. Chapter 5 lists those extensions to the language (besides parameters and resources) since the publication of the *PostScript Language Reference Manual, Second Edition*.

1.1.4 Compatibility Operators

For compatibility with existing Level 1 PostScript language driver software, which might depend on operators that were often present in PostScript Level 1 products, a collection of compatibility operators and procedures is included in each Level 2 implementation. These compatibility operators are described in chapter 6.

1.2 Terminology Used in This Manual

Throughout this manual, the following terms are used.

- *Device*. A device is defined as a piece of hardware under the control of a PostScript interpreter. There are several categories of devices:

  Page device. A page device can be, for example, a laser print engine producing paper output.

  Communication device. A communication device can be, for example, serial, parallel, or LocalTalk communications hardware and software.

  File system device. A file system device can be, for example, a disk or cartridge system.

- *Host*. A host is defined as a computer system (for example, a personal computer or workstation) connected to a PostScript printer product via one of its communications devices. The host sends PostScript language programs over the communications channel to the printer. The printer executes them.

- *PostScript interpreter.* A PostScript interpreter is defined as a body of software that executes programs written in the PostScript language and produces effects such as generating printed output on a page device.

- *PostScript language product.* A PostScript language product is defined as a system consisting of a PostScript interpreter controlling one or more devices.

1.3 Related Publications

Adobe Communications Protocols Specification, available both from the Adobe Developers Association and in the Adobe Printer Development Kit (PDK) describes several protocols that can be used to communicate over a serial or parallel connection to a PostScript language printing device.

PostScript Language Program Design (Reading, MA: Addison-Wesley, 1988) teaches programming principles unique to the Level 1 PostScript language and contains many usable samples. It is for programmers interested in the effective and efficient design of PostScript language programs and printer drivers.

PostScript Language Reference Manual, Second Edition (Reading, MA: Addison-Wesley, 1990) is the programmer's reference manual for the PostScript language. It describes the syntax and semantics of the language, the imaging model and the effects of the graphical operators.

CHAPTER 2

Page Device Parameters

This chapter describes the new page device parameters that have been added or modified since the publication of the *PostScript Language Reference Manual, Second Edition*. These parameters are part of a dictionary that is used to model the internal state of a page device, which is usually a raster output device that produces a sequence of pages on physical media such as sheets of paper. Two operators, **currentpagedevice** and **setpagedevice**, respectively, read and set the parameter values.

For more information about the PostScript language facilities for setting up a raster output device, refer to section 4.11, “Device Setup,” in the *PostScript Language Reference Manual, Second Edition*.

The following page device parameters are described in the *PostScript Language Reference Manual, Second Edition*. The description of these parameters is unchanged.

AdvanceDistance	**AdvanceMedia**	**BeginPage**
Collate	**CutMedia**	**Duplex**
EndPage	**HWResolution**	**ImagingBBox**
Install	**ManualFeed**	**MediaColor**
MediaType	**MediaWeight**	**MirrorPrint**
NegativePrint	**NumCopies**	**Orientation**
OutputAttributes	**OutputFaceUp**	**PageSize**
Policies	**Separations**	**Tumble**

Table 2.1 on page 6 describes the page device parameters that have been defined or amended since the publication of the *PostScript Language Reference Manual, Second Edition*.

Section 2.1 on page 18 explains the semantics of the **Details** dictionaries.

Section 2.2 on page 20 describes the **DeviceRenderingInfo** dictionaries.

Section 2.3 on page 21 explains the PostScript language fax interface and describes the **FaxOptions** dictionary keys.

Section 2.4 on page 39 explains envelope orientation in user space.

Section 2.5 on page 40 describes errors generated by page device parameters.

Table 2.1 *Page device parameters*

Key	*Type*	*Semantics*
Bind	integer	This parameter requests that the document be bound. The document will be bound at a specific time indicated by an integer code: 0 Do not bind. 1 Bind at device deactivation. 2 Bind at the end of the job. 3 Bind after each set. 4 Bind after each **showpage** or **copypage**. See the descriptions of **Collate** and **Jog** in section 4.11.3 of the *PostScript Language Reference Manual, Second Edition*, for a description of the above terminology.
BindDetails	dictionary	This dictionary describes product-specific details related to how a document is to be bound. For more information on **Details** dictionaries, see section 2.1, "Details Dictionaries."
Booklet	boolean	This parameter requests that the document be stapled, trimmed and folded into booklet form.
BookletDetails	dictionary	This dictionary describes product-specific details related to how a document is to be stapled, trimmed and folded. For more information on **Details** dictionaries, see section 2.1, "Details Dictionaries."
CollateDetails	dictionary	This dictionary describes product-specific details related to how a document is to be collated. For more information on **Details** dictionaries, see section 2.1, "Details Dictionaries."
DeferredMediaSelection	boolean	This page device key is found on those devices that support two different media models. A value of *false* indicates that media selection is to be done as described in section 4.11.4 of the *PostScript Language Reference Manual, Second Edition*. A value of *true* indicates that the product will be completely responsible for verifying the media requests. The motivation for this is usually that the product is handing the requests off to another "entity" that will guarantee the media requests will be satisfied at printing time. This is the reason for the name **DeferredMediaSelection**—the actual selection of the media is deferred to a time later than the time of this execution of **setpagedevice**.

Note *When **DeferredMediaSelection** is true and the **PageSize** is rejected by a product, **PageSize** policies 0 and 3 through 6 will result in a configuration error. Also, if **PageSize** policy 7 is being used and **DeferredMediaSelection** is true, it is product-dependent as to whether or not media will be pulled from the current tray.*

When **DeferredMediaSelection** transitions between *true* and *false*, some media parameters may need to be initialized again. For example, each media selection model may have its own notion of the default page size that should be used if **PageSize** has not been explicitly specified.

DeviceRenderingInfo
dictionary This dictionary provides a location for individual OEMs or products to specify their own device rendering parameters. The only required key is **Type** of type integer. When **setpagedevice** is executed, the entries in the **DeviceRenderingInfo** dictionary will be checked only if the **Type** value matches the value expected by the printing device. If the value does not match, the **setpagedevice** implementation will consult **Policies**. If the **Type** key is not specified in the **DeviceRenderingInfo** dictionary, an **undefined** error will result. (See also section 2.2, "DeviceRenderingInfo Dictionaries.)

ExitJamRecovery boolean If *true*, pages that jam in the exit path are reprinted. If *false* (jam recovery disabled), performance might be improved because more overlapping of page processing is possible.

FaxOptions dictionary This dictionary provides a number of ways of customizing fax jobs. Information in the dictionary is placed on cover sheets and transmission reports and is recorded in logs. Procedures in the dictionary are used for page captions, cover sheets and transmission reports. Other values in the dictionary control broadcast, delayed transmission and high-resolution capabilities.

For a description of the **FaxOptions** key and the fax interface, see section 2.3, "PostScript Language Interface to Fax."

Fold integer This parameter requests that the document be folded. The document will be folded at a specific time indicated by an integer code:

0	Do not fold.
1	Fold at device deactivation.
2	Fold at the end of the job.
3	Fold after each set.
4	Fold after each **showpage** or **copypage**.

See the descriptions of **Collate** and **Jog** in section 4.11.3 of the *PostScript Language Reference Manual, Second Edition*, for a description of the above terminology.

FoldDetails dictionary This dictionary describes product-specific details related to how a document is to be folded. For more information on **Details** dictionaries, see section 2.1, “Details Dictionaries.”

ImageShift array This array, [x y], specifies the distance, in default user space, that each image on a page is to be shifted in the *x* direction for *x* units and in the *y* direction for *y* units. For page images that are to appear on a front side, the horizontal shift is to the right if $x > 0$, or to the left if $x < 0$. The vertical shift is to the top if $y > 0$, or to the bottom if $y < 0$. For page images that are to appear on a back side, the horizontal shift is to the left if $x > 0$, or to the right if $x < 0$. The vertical shift is to the bottom if $y > 0$, or the top if $y < 0$.

InputAttributes dictionary If **InputAttributes** is *null* instead of a dictionary, the PostScript interpreter has no previous knowledge of the available media. When **setpagedevice** is executed, the interpreter simply presents media selection requests to the device implementation, which is fully responsible for determining if they can be satisfied. This arrangement exists in products where actual printing of the output is deferred to some process not directly under the control of the PostScript interpreter.

The **InputAttributes** dictionary normally contains an entry for each input media source. The entry consists of an integer representing the input paper slot and an associated dictionary. Some products may support the **InsertSheet** boolean entry in the slot dictionaries that are subdictionaries of **InputAttributes**. This entry indicates whether or not the slot holds special insert sheet media. **InsertSheet** is used during the media matching process and compared against the **setpagedevice** key **InsertSheet** (described next).

InsertSheet boolean This parameter specifies whether or not to select inserted media. **setpagedevice** compares it with the **InsertSheet** value, if any, in the **InputAttributes** entries for all media that it considers. Refer to section 4.11.4, “Media Selection,” in the *PostScript Language Reference Manual, Second Edition*, keeping in mind that **InsertSheet** is also an input media entry found within the **InputAttributes** dictionary.

A side-effect of executing **setpagedevice** with **InsertSheet** equal to *true* and selecting an insert sheet slot is that the imageable area gets set to a zero-area region to ensure that nothing is imaged on the inserted sheet. That is, the insert sheet is explicitly not imaged. The insert sheet slot has the special property that the media coming from this slot is not sent through the fuser (a hot device in laser printers that fuses the toner to the paper) on its way to the output bin. The media pulled from an **InsertSheet** slot does not go through the normal paper path. Photographic material is a good example of **InsertSheet** media which has the special requirement that it cannot tolerate being imaged to, nor sent through the fuser, without major damage.

Here is an example of how to use **InsertSheet**:

```
%... PostScript language code for page n
%... page n+1 is an inserted sheet
%
    save
    <</InsertSheet true>> setpagedevice
      % selects InsertSheet media
    showpage
      %send the InsertSheet media on to the
      %output bin as page n+1
    restore
      %implicitly go back to using the regular media
%
%...PostScript language code for page n+2
```

Jog integer Requests that output pages be "jogged" — physically shifted in the output bin or diverted to another output bin — at specific times indicated by an integer code. Changes to the **Jog** parameter take effect only after all pages prior to the current page description containing the requested changes have been delivered or aborted. The flag is acted upon and reset only after all pages preceding the jog trigger have been delivered or aborted. Multiple jog requests with no intervening pages act like a single jog request. The request codes are:

0 Do not jog pages at all.

1 Jog at device deactivation. The notion of "device deactivation" is explained in section 4.11.6, "BeginPage and Endpage," of the *PostScript Language Reference Manual, Second Edition*. A jog request is triggered when the **Jog** parameter in the deactivating device has the value 1.

The following is an illustration of the **Jog** option 1:

```
<< /Jog 0 >> setpagedevice
showpage   % page 1
<< /Jog 1>> setpagedevice
showpage   % page 2
<< /Jog 0>> setpagedevice
showpage   % page 3
```

Jogging will occur before page 3 is printed; pages 2 and 3 will be separated in the output bin.

2 Jog at the end of the job. The notion of a "job" is explained in section 3.7.7, "Job Execution," of the *PostScript Language Reference Manual, Second Editio*n. Jogging between jobs is controlled by the value of **Jog** for the page device that is

current between jobs. Thus, this feature can be turned on or off only by executing **setpagedevice** as part of an unencapsulated job.

In the following illustration of multiple **Jog** requests without intervening pages being printed, assume that jogging between jobs is in effect (**Jog** option 2 was set in an unencapsulated job):

```
showpage          % Job, page 1
^D                % EOJ
(Job 2 prints no pages)
^D                % EOJ
showpage          % Job 3, page 2
^D                % EOJ
```

Although jog requests occur at the end of jobs 1 and 2, only a single **Jog** action will occur before page 2 is printed.

3 — Jog before each set and at device deactivation. The notion of a "set" is explained in the description of the **Collate** entry.

Laminate boolean — If *true*, the page is to be laminated. If *false*, the page is not to be laminated. How a page is laminated is product-specific.

LeadingEdge

integer or null — This parameter specifies the edge of the media that will enter the engine or imager first and across which data will be imaged. **setpagedevice** compares the value of **LeadingEdge** (unless its value is *null*) with the values of **LeadingEdge** in the **InputAttributes** entries for all defined media. Refer to section 4.11.4, "Media Selection," in the *PostScript Language Reference Manual, Second Edition.*

Values of **LeadingEdge** reflect positions relative to a canonical page, the orientation of which matches the default user space that would be selected by **PageSize** [x,y] **Orientation** 0, where *x* and *y* are the dimensions of the media, with $x < y$. Possible values are as follows:

null — No request for media orientation.

0 — Short edge; top of canonical page.

1 — Long edge; right side of canonical page.

2 — Short edge; bottom of canonical page.

3 — Long edge; left side of canonical page.

When duplex printing is enabled, the canonical page orientation describes only the front side of the media. The orientation of the reverse side is defined by **Tumble** and is independent of the value of **LeadingEdge**.

ManualFeedTimeout

integer — The number of seconds the printer waits for a page to be fed manually before generating a time-out error. A zero value means no time-out (infinite wait).

Margins array — If the device supports multiple resolutions (that is, different values of **HWResolution**), the margin values are interpreted according to some canonical default resolution and are scaled appropriately at other resolutions. This ensures that they represent the same physical distance when the resolution is varied. The canonical default resolution is product-dependent and specified in the *Addendum* for the product. For more information on **Margins**, see Table 4.11 in section 4.11.3 of the *PostScript Language Reference Manual, Second Edition*.

MediaClass string or null — Specifies the class of media. If **MediaClass** is not *null*, **setpagedevice** compares this value with the **MediaClass** values, if any, in the **InputAttributes** entries for all media that it considers. Refer to section 4.11.4 of the *PostScript Language Reference Manual, Second Edition*. The value of **MediaClass** is product-specific. Declare the set of values in the **OutputDevice** resource. (This resource entry is not required if the product does not support the **MediaClass** key.) See also **MediaType**, which is an arbitrary string for media selection, in Table 4.10 of the *PostScript Language Reference Manual, Second Edition*.

A product should support this key when the media selected for the output requires some action that may affect the output; for example, color rendering dictionary (CRD) selection or paper/transparency selection. See also the **PageDeviceName** page device parameter for CRD selection.

MediaPosition

integer or null — This is an integer that indicates the slot that is to be used. The interpretation of the integer is product-specific since slot numbers themselves are product-specific. If media matching is in effect, **MediaPosition** does not override the matching process but does alter it (making use of policy whenever possible) so that if the requested slot number can be chosen in a manner that is consistent with media matching, it will be selected. Note that, consequently, if the slot specified by **MediaPosition** is selected, that slot is not necessarily the best match for the page device requests. If the requested slot is not installed or inserted, or if it cannot be chosen in the manner described above, **MediaPosition** will be rejected according to its policy.

For example, consider a printer with two slots, slot 0 containing legal paper and slot 1 containing letter-size paper. The policies of **PageSize** and **MediaPosition** are both 1. The command

```
<</PageSize [612 1008] /MediaPosition 1>> setpagedevice
```

would result in slot 1 (with letter-size paper) being selected, even though slot 0 is the perfect match for the request page size. This is because slot 1, as specified by **MediaPosition**, can satisfy the page device request (by ignoring the **PageSize** request) and so this slot is chosen.

The same result holds if the policy of **PageSize** is 3. Once again, this is because slot 1 can satisfy the page device request (by scaling and adjusting the page).

If the policy of **PageSize** is 0, then slot 1 cannot satisfy the page device request (in the sense that **setpagedevice** will raise a configuration error). In this case, **MediaPosition** will be rejected. Since its policy is 1 in this example, its value is ignored, and media matching will be retried on all slots. The result is that slot 0 (containing legal paper) is chosen.

If media matching is not in effect (for example, **DeferredMediaSelection** is supported and *true*) then it is a product decision as to how to resolve potential conflicts between the various media requests and the **MediaPosition** request. If **MediaPosition** is set to *null*, it plays no role in the media selection process. If the **PageSize** policy is 7 or if manually feeding, **MediaPosition** is ignored.

OutputDevice

name or string

This parameter selects an output device in environments in which the PostScript interpreter can generate output for multiple page devices. In some environments, it selects among different types of output devices, such as a printer and a fax modem, a printer and a display screen, or a printer and an imagesetter. In other environments, it may select among similar devices, such as two or more imagesetters.

When the value of **OutputDevice** changes, the usual inheritance of values not specified in the operand to **setpagedevice** does not happen. Instead, all new values are generated in a manner that is specific to each product. Also, the set of acceptable keys for **setpagedevice** can change when changing the value of **OutputDevice**, since different devices have different features that can be controlled or queried.

OutputPage boolean

If *true*, processing is normal. If *false*, no pages are actually printed, but all other processing is done as if the page were to be printed, including rasterizing to a frame buffer. Thus, when **OutputPage** is *false*, the time to process a page includes everything except time spent waiting for the marking engine.

Furthermore, rasterization occurs synchronously with execution of **showpage** instead of being overlapped with processing of subsequent pages. This facilitates measuring the complete cost of page execution.

PageDeviceName

string, name or null — This parameter is used by the **findcolorrendering** operator and provides a way to label a specific device setup. Refer to page 176 for details about the **findcolorrendering** operator. See also **MediaClass**, which can affect CRD selection.

PageOffset array — This array, [x y], contains two numbers that are used to relocate the page image on the media *x* units in the device *x* coordinate direction and *y* units in the device *y* coordinate direction. *x* and *y* are always expressed in units of 1/72 of an inch. This positioning is typically accomplished by altering the current transformation matrix (CTM). However, on some products, this positioning can be accomplished by device-dependent means that are independent of the graphics state (the CTM in particular). The **PageOffset** key is typically found on imagesetters and is used to control where the image is to appear on the media.

PageSize array — Refer to Table 4.10 on page 232 in the *PostScript Language Reference Manual, Second Edition* for a description of **PageSize**. Further clarification is needed, though, regarding how the **PageSize** matching tolerance is used during the media matching process.

When using roll media, the media is considered to be a match (with respect to the **PageSize**) when the requested media size is less than or equal to the actual size of the physical media plus 5 additional default user space units. Hence, even if a match succeeds, the requested dimensions may in fact be larger than the actual dimensions of the physical media.

For non-roll media, the requested size must lie between the actual size minus 5 default user space units and the actual size plus 5 units in order to be considered a match (in other words, the absolute difference between the actual and request sizes is no more than 5 units). When non-roll media is matched, the dimensions used are the actual dimensions of the actual media selected. Failure to match any available media within this tolerance triggers the **PageSize** recovery policy in either case.

PostRenderingEnhance

boolean — If *true*, product-specific image enhancements are enabled. These enhancements are made *after* the page is rasterized in memory.

PostRenderingEnhanceDetails

dictionary — This dictionary describes product-specific details related to the post-rendering image enhancement. For more information on **Details** dictionaries, see section 2.1, “Details Dictionaries.”

PreRenderingEnhance

boolean — If *true*, product-specific image enhancements are enabled. These enhancements are made *before* the image is rasterized in memory.

PreRenderingEnhanceDetails

dictionary — This dictionary describes product-specific details related to prerendering image enhancement. For more information on **Details** dictionaries, see section 2.1, “Details Dictionaries.”

ProcessColorModel

name or string — This name or string value specifies the colorant model used for rendering process colors in the device. It affects rendering for all color spaces, with the exception of **Separation** color spaces that actually produce separations. It does not affect the interpretation of color values in any color space; it controls only the rendering method.

Legal values are /DeviceGray, /DeviceRGB, /DeviceCMYK, /DeviceCMY and /DeviceRGBK. For example, /DeviceRGB specifies that the process colorants are named red, green and blue; /DeviceCMYK specifies cyan, magenta, yellow and black. These are the process colorant names used to select halftones in a Type 5 halftone dictionary and to control the production of separations in **SeparationColorNames** and **SeparationOrder**.

Each of the **ProcessColorModel** values implies a specific native color space for the device. The native color space is the PostScript language device color space into which user-specified colors are converted if necessary; see section 6.2 in the *PostScript Language Reference Manual, Second Edition*.

- /DeviceGray, /DeviceRGB and /DeviceCMYK select the correspondingly named native device color space.
- /DeviceCMY and /DeviceRGBK both select /DeviceRGB as the native device color space, but they cause the device to render the /DeviceRGB color values in special ways. For /DeviceCMY, the device renders the RGB colors using the complementary subtractive colors. For /DeviceRGBK, the device uses a separate rendering method for RGB color values that represent pure shades of gray.

RollFedMedia boolean — If the value of **RollFedMedia** is *true*, the media is roll fed. Note that the matching criterion for the **PageSize** parameter differs for roll-fed and non-roll-fed media (see the **PageSize** page device parameter in this table.)

SeparationColorNames

array — This parameter specifies those **Separation** color spaces that the device supports. This array can contain either names or strings, for example, [/ Pink /Green] or [(Pink) (Green)] or [/ Pink (Green)].

If the name used in a [/Separation name ...] **setcolorspace** operation is included in this array, that colorant will be used, rather than the alternate color space. Any other color will be mapped to one or more of the named colors through the alternate color space and tintTransform parameter of the **setcolorspace** operator. This is described in section 4.8.4 of the *PostScript Language Reference Manual, Second Edition*.

The names of the colorants of the native color space are included implicitly, regardless of the contents of the array. Thus:

- for /DeviceCMY, the empty array [] is equivalent to [/Cyan /Magenta /Yellow].
- for /DeviceCMYK, the empty array [] is equivalent to [/Cyan /Magenta /Yellow /Black].
- for /DeviceRGB, the empty array [] is equivalent to [/Red /Green /Blue].
- for /DeviceRGBK, the empty array [] is equivalent to [/Red /Green /Blue /Black].
- for /DeviceGray, the empty array [] is equivalent to [/Gray].

SeparationOrder array If separations are being made, this parameter specifies that they be produced in the order given by the array of color names (where an array can contain either names or strings, such as [/Cyan /Magenta] or [(Cyan) (Magenta)]). Legal values are the names of the colorants of the native color space, as well as any additional names specified by **SeparationColorNames**.

A separation will be produced for each occurrence of a name; multiple occurrences will produce multiple separations. No separations will be produced for colors whose names are not given, regardless of their appearance in **SeparationColorNames**. The named separation color space is defined (as opposed to reverting to the alternative color space), but the output for that separation is discarded when a certain color name is not given.

An empty array [ ] requests that separations for all colors of the native color space, as well as all colors requested by **SeparationColorNames**, be produced in an unspecified order.

When not making separations, some devices may use **SeparationOrder** to determine the colorants and the order in which they are to be applied to the composite image.

Signature boolean If *true*, the job will be "signatured." That is, pages of a document will be arranged so that, when folded, the pages will be in the right order. How signaturing is performed is device-dependent. On some devices, the engine may provide the resources (memory, disk space) to signature the job. On other devices, the interpreter may have to reorder the virtual pages in order to deliver the pages to the engine in the correct order. In the latter case, a **Signature** value of *true* implies that the interpreter must store the results of executing the page description for multiple pages in order to deliver the pages correctly ordered. This use of **Signature** is supported by relatively few products and is subject to resource limits in products that do support it.

SlipSheet integer This parameter requests that slip sheets (slip sheet media selection is product-specific) be inserted. There is no way to render a slip sheet (the imageable area gets set to a zero-area region); the engine is simply told when to insert it. For example, a slip sheet can be a colored sheet of paper that visually separates multiple copies. A slip sheet may go through the fuser or a separate paper path. Compare with the description of **InsertSheet** above.

Slip sheets will be inserted at specific times indicated by an integer code:

0	Do not insert slip sheets.
1	Insert slip sheet at device deactivation.
2	Insert slip sheet at the end of the job.
3	Insert slip sheet at the end of the set.
4	Insert slip sheet after each **showpage** or **copypage**.

See the descriptions of **Collate** and **Jog** in section 4.11.3 of the *PostScript Language Reference Manual, Second Edition*, for a description of the above terminology.

Staple integer This parameter requests that the job be stapled. The job will be stapled at a specific time indicated by an integer code:

0	Do not staple.
1	Staple at device deactivation.
2	Staple at the end of the job.
3	Staple after each set.
4	Staple after each **showpage** or **copypage**.

See the descriptions of **Collate** and **Jog** in section 4.11.3 of the *PostScript Language Reference Manual, Second Edition*, for a description of the above terminology.

StapleDetails dictionary This dictionary describes product-specific details related to how a document is to be stapled. For more information on **Details** dictionaries, see section 2.1, "Details Dictionaries."

TraySwitch	boolean	If *true*, automatic tray switching is provided. This option is offered by some devices with multiple input trays. When one input tray runs out of media, another tray with the same type of media can be used automatically, without alerting you that the printer is out of media. Tray switching can take place between any of a printer's trays as long as they have the same characteristics (for example, **PageSize**, **MediaColor** and so forth) as the selected tray. These alternative trays and their priorities are a product-dependent feature. They do not depend on the **Priority** array; however, the **Priority** array may be used for this purpose.
Trim	integer	This parameter requests that the job be trimmed. The job will be trimmed at a specific time indicated by an integer code: 0 Do not trim. 1 Trim at device deactivation. 2 Trim at the end of the job. 3 Trim after each set. 4 Trim after each **showpage** or **copypage**. See the descriptions of Collate and Jog in section 4.11.3 of the *PostScript Language Reference Manual, Second Edition*, for a description of the above terminology.

Table 2.2 describes changes to the **PageSize** entry in the **Policies** page device dictionary.

Table 2.2 *PageSize entry in the Policies dictionary*

Key	*Type*	*Semantics*
PageSize	integer	This parameter specifies the recovery policy to use when the **PageSize** cannot be matched (within a tolerance of 5 units) with any available media. The policy values 0-6 are described in Table 4.14 in section 4.11.5, "Policies," in the *PostScript Language Reference Manual, Second Edition*. The new policy value 7 has been added since the publication of the manual. 7 Disable media selection altogether and impose the requested **PageSize** on the previously selected medium without adjusting it in any way. That is, set up the page device as if the selected medium were of the requested size, ignoring the medium's actual size. However, if the requested **PageSize** is not within 5 units of any one of the page sizes supported by the product, the product should reject **PageSize** with policy 7 and a configuration error will result. The manner in which the page image will be positioned on the medium is product-dependent and unpredictable.

When the **PageSize** policy is 7, it takes effect during every execution of **setpagedevice**. This is unlike all other policies, which take effect only if a request cannot be satisfied.

This policy exists solely for use in the emulations of certain Level 1 compatibility operators that perform media selection and page device setup separately. **PageSize** policy 7 should never be used in a Level 2 application. Its semantics violate the Level 2 page device model, and documents using it are not portable.

2.1 Details Dictionaries

Certain page device features have many variables which determine precisely how the feature functions; these variables may be quite different on different products. Such a feature is enabled or disabled by a primary page device entry, while the exact way in which the feature functions is determined by secondary entries in a **Details** dictionary page device entry. This allows an application that is not knowledgeable about the details of the feature to enable and disable it, while more sophisticated utilities can be used to configure the details separately.

An example of this is the stapling feature. Many applications will want to either enable or disable stapling with the assumption that the number, location and orientation of the staples has been configured correctly. The nature of the configuration will be dependent on the printing device. For example, for some engines it may be possible to specify an arbitrary staple location on the sheet, while on others, staples may be placed only in the four corners.

Primary page device entries for such features are either booleans or integers. If the value is a boolean, then the feature is enabled if the value is *true* and disabled if the value is *false*. If the value is an integer, the feature is disabled if the value is zero. The non-zero values enable the feature in different ways that are consistent across all products. For example, the binding feature can be enabled for binding at the end of device deactivation, at the end of a job, at the end of each set or at each **showpage** or **copypage**.

A consistent naming convention is used for **Details** dictionaries. The name of the dictionary is the name of the primary key with **“Details**” appended. For example, if the **Staple** feature is present and has a details dictionary, this dictionary is named **StapleDetails**.

A **Details** dictionary will be present for a given feature on a given product only if additional information beyond that of the primary entry is needed to control it. For example, a product supporting a postrendering enhancement feature that can only be enabled or disabled without further control will not

have a **Details** dictionary for this feature. Such a **Details** dictionary would be present on a printer with more configurable postrendering enhancement. Applications which are simply enabling and disabling a feature should never reference a **Details** dictionary. More sophisticated applications intentioned to control a **Details** dictionary should never assume that one is present unless the exact nature of the printing device on which they are executing is known.

During the execution of **setpagedevice**, the entries in any **Details** dictionary are checked to be syntactically correct only if the **Type** value matches what is expected by the printing device. If the **Type** key is not specified in the **Details** dictionary, an **undefined** error will result. When the **Type** value is a number not known by the printing device, policy is consulted. When the **Type** value is known, the validity of the values within the **Details** dictionary are only checked if the feature will be enabled for the page device in effect as a result of **setpagedevice**. As with all page device entries, syntactically incorrect settings result in appropriate PostScript language errors (for example, **typecheck**) and invalid values result in policy being consulted.

The Type Entry

Every **Details** dictionary has a **Type** entry whose integer value completely determines how the **Details** dictionary entries affect the feature. That is, if two different products have **Details** dictionaries for the same feature and the **Type** entry is the same for each, then the dictionaries will have exactly the same named entries and the syntax and semantics of each entry will be the same. This allows an application, based solely on the value of the **Type** entry, to change entries in a **Details** dictionary for a feature.

If **Details** dictionary entries are being set, whether the new dictionary overwrites the current one or is merged with it, is determined by the **Type** entry. The criteria for merging versus overwriting is product-dependent. **Details** dictionaries and their associated **Type** entries are registered by Adobe.

Table 2.3 describes some of the **CollateDetails** keys being used in products; a complete description of available keys for a given product will be provided by the OEM.

Table 2.3 *CollateDetails parameters*

Key	*Type*	*Semantics*
ProofSet	boolean	If *true*, the initial copy of a collated job will be printed before remaining copies are output. The printer will remain in a “hold” mode until some product-specific action is taken, such as aborting or printing the remaining copies.

Type	integer	Identifies the type of **CollateDetails** dictionary. A given type of **CollateDetails** dictionary can contain a specific set of keys whose values are interpreted in a particular way. Different dictionary types are independent, and a given product supports only certain specific types.

Table 2.4 describes some of the **PostRenderingEnhanceDetails** keys being used in products; a complete description of available keys for a given product will be provided by the OEM.

Table 2.4 *PostRenderingEnhanceDetails parameters*

Key	*Type*	*Semantics*
REValue	integer	This parameter provides a range of values for rendering enhancement. A value of zero reflects the least (or no) amount of enhancement.
PrintQuality	integer	This parameter provides a range of output quality levels as a positive integer. A value of zero reflects the lowest or draft quality.
Type	integer	This parameter identifies the type of **PostRenderingEnhanceDetails** dictionary. A given type of **PostRenderingEnhanceDetails** dictionary can contain a specific set of keys whose values are interpreted in a particular way. Different dictionary types are independent, and a given product supports only certain specific types.

2.2 DeviceRenderingInfo Dictionaries

Certain products must be able to specify their own device rendering parameters, and these parameters may be quite different on different products. Unlike **Details** dictionaries, these parameters are applied as needed for rendering to occur and are not disabled by a primary page device entry.

During the execution of **setpagedevice**, the entries in the **DeviceRenderingInfo** dictionary are checked for syntactical correctness only if the **Type** value matches what is expected by the printing device. If the **Type** key is not specified in the **DeviceRenderingInfo** dictionary, an **undefined** error will result.When the **Type** value is a number not known by the printing device, policy is consulted. As with all page device entries, syntactically incorrect settings generate PostScript language errors (for example, **typecheck**) and invalid values result in policy being consulted.

The Type Entry

Every **DeviceRenderingInfo** dictionary has a **Type** entry whose integer value determines how the dictionary entries affect the rendering capability. Thus, if two different products have **DeviceRendeirngInfo** dictionaries with the same **Type**, then the dictionaries have exactly the same named entries, and the

syntax and semantics of each entry will be the same. This allows an application, based solely on the value of the **Type** entry, to change entries in a **DeviceRenderingInfo** dictionary for the rendering capability.

Table 2.5 describes some of the **DeviceRenderingInfo** keys being used in products; a complete description of available keys for a given product will be provided by the OEM.

Table 2.5 *DeviceRenderingInfo parameters*

Key	*Type*	*Semantics*
ValuesPerColorComponent	integer	This parameter specifies the number of values each color component may have, or in the monochrome case, the number of gray levels.
AntiAlias	boolean	When *true*, this parameter enables the antialiasing feature where available. Currently this requires **ValuesPerColorComponent** to be greater than 1 and for **ColorBurst**™ to be present.
Type	integer	Identifies the type of **DeviceRenderingInfo** dictionary. A given type of **DeviceRenderingInfo** dictionary can contain a specific set of keys whose values are interpreted in a particular way. Different dictionary types are independent, and a given product supports only specific types.

2.3 PostScript Language Interface to Fax

This section is intended for use by programmers who are building PostScript language drivers and utility software to work with PostScript fax printers. PostScript language drivers create pages to be printed and sent via fax, and consequently need to know how to create PostScript fax jobs in addition to PostScript language print jobs.

PostScript fax printers require utility software on the host computer to set variables in the printer, such as time-of-day, number of rings to answer on, speaker settings, and so on. These settings persist across multiple fax jobs and multiple users. Programmers developing utility software need a language interface to read and write this data.

The description of the PostScript language interface to fax is broken down into two major sections.

- “PostScript Language Interface to Fax,” section 2.3, describes the interface which controls individual fax jobs. “Some Sample Fax Jobs,” section 2.3.7 presents several example raster and PostScript fax jobs.

- "The Fax Environment Interface," section 3.6, describes the device parameters and administrative resources needed to set, control and examine the fax environment shared by all jobs and all users connected to the fax printer.

2.3.1 The Fax Job Interface

PostScript fax printers may send faxes in two different ways: they may send raster images, and they may send PostScript language files. Raster images may be received by any ITU (International Telecommunications Union) Group 3 fax machine. PostScript language files may only be received by suitably equipped machines. The PostScript language interfaces for these two different, but related, methods are described in sections 2.3.2, "Sending Raster Images," and 2.3.3, "Transmitting PostScript Language Files." Common to both is a collection of dictionary keys which contain information, such as telephone numbers, necessary to carry out the fax operation. These keys are described in section 2.3.4, "The Fax Options Dictionary Keys."

Cover sheets, transmission reports and page captions are traditional elements of fax communication. The job interface makes it possible to generate these items. This is described in section 2.3.5, "CoverSheet, Confirmation and PageCaption Procedures."

Fax Job Transmissions

Fax job transmissions can be made to single or multiple destinations.

A fax transmission in raster form to a single destination will be made in a single call if there is sufficient storage to hold all of the raster pages of the job. However, the transmission may be broken into multiple calls if there is not sufficient storage. If the storage for the raster pages is exhausted, then a call will be made before rasterization is complete. Transmission and rasterization will then take place simultaneously. If transmission succeeds in sending all prepared pages before rasterization of the job is finished, the call will end. A subsequent call will be made either when all of the rasterization is done or when storage is again exhausted. Thus, a transmission may be broken into multiple calls. It is also possible that when transmission and rasterization are taking place simultaneously, rasterization keeps up with transmission. In this case, this transmission will be the last call for the job.

If the storage in use is disk storage, there will typically be sufficient storage for all of the raster pages of a job and the fax will be transmitted in a single call.

It is possible to request that a transmission take place at a particular time. These delayed transmissions, of course, are made with a single call. Since the pages of a raster job are prepared when the job is submitted, all of the pages of a delayed raster transmission must fit in storage otherwise the job will fail.

The transmission of a PostScript language file to a single destination is always made in a single call. This call takes place only after the entire file is in storage. If the file will not all fit in storage, the job will fail.

The transmission of raster files and PostScript language files to multiple destinations is described in “Broadcast Transmission of Faxes,” section 2.3.6.

2.3.2 Sending Raster Images

PostScript language files can be sent as raster images to ITU Group 3 fax machines by selecting /Fax as the current page device with the **setpagedevice** operator:

```
<</OutputDevice /Fax
other key-value pairs >> setpagedevice
```

With a call to **setpagedevice**, one of three cases is possible:

1. A new page device is established that does not change the value of **OutputDevice** in the current page device. In this case, entries in the new page device which are not specified in the argument dictionary to **setpagedevice** are inherited from the current page device.

2. A new page device is established when there is no current page device. In this case, entries in the new page device which are not specified in the argument dictionary to **setpagedevice** are initialized to product-dependent default values.

3. A new page device is established that changes the value of **OutputDevice** in the current page device. In this case, entries in the new page device are not inherited from the current page device; rather, the entries in the new page device which are not specified in the argument dictionary to **setpagedevice** are initialized to product-dependent default values.

When the **OutputDevice** is /Fax, one of the entries in the page device dictionary is the **FaxOptions** dictionary. Values in the **FaxOptions** dictionary, if any, supplied as part of the argument dictionary to **setpagedevice** are merged to one level into the **FaxOptions** dictionary held in the page device. This is similar to the treatment of **InputAttributes** and **OutputAttributes**. Section 2.3.4, “The Fax Options Dictionary Keys,” describes the entries found in the **FaxOptions** dictionary and also lists typical default values.

If the call to the **setpagedevice** operator does not change the **OutputDevice**, (case 1 above) then the **FaxOptions** dictionary before and after the merge of the argument dictionary is considered. If the **FaxOptions** dictionary has been changed in any way, then it is assumed that this is a new transmission; the previous one is finished (cover sheets are generated, the phone call is queued, and so forth) and a new transmission context is started. The reliable technique is to establish all the **FaxOptions** during the first call to **setpagedevice** that has the **OutputDevice** set to /Fax.

The FaxOptions DialCallee Key

One of the entries in the **FaxOptions** dictionary, the **DialCallee** key, contains the phone number for the fax machine to call. If the **DialCallee** key has a *null* value, the fax transmission cannot proceed, and a **typecheck** error will be raised. The **DialCallee** key is described in greater detail in section 2.3.4.

If entries in the new page device are initialized to product-dependent default values (see cases 2 and 3 above), then the argument dictionary must contain an entry for **FaxOptions**, which in turn must contain a value for the **DialCallee** key. If the **FaxOptions** dictionary does not contain a value for the **DialCallee**, it will be assigned the product-dependent default value (a *null*). Fax transmission cannot proceed using a *null* value as the phone number to call and a **configurationerror** will be raised.

2.3.3 Transmitting PostScript Language Files

It is possible to transmit jobs as PostScript language files between consenting PostScript fax printers. A job may be sent in PostScript language form by using the **faxsendps** operator found in the **FaxOps ProcSet** instance:

```
file OptionsDict
/FaxOps /ProcSet findresource /faxsendps get exec
```

OptionsDict is an argument dictionary consisting of a set of **FaxOptions** keys. Two entries in this argument dictionary control the transmission of jobs as PostScript language files. These are **RevertToRaster** and **PostScriptPassword**. The %Fax% device's **ReceivePostScript** and **PostScriptPassword** parameters (see Table 3.41 on page 143) control the receipt of PostScript language jobs by a fax printer.

file is typically **currentfile**, representing the remainder of the current job. The **faxsendps** operator will either send this file to the remote printer to be executed as a PostScript language job, or execute it locally producing a raster fax. The following four steps describe how this is done.

1. The file is read with the contents being saved internally. This continues until end-of-file (EOF) is reached or until fax storage is all used up.

   If there is not sufficient storage to hold the entire file, the **RevertToRaster** key in the *OptionsDict* dictionary is consulted. If **RevertToRaster** is *false*, the job fails and the message "`Storage to assemble PostScript transmission exhausted`" will be sent to %stdout. If it is *true*, then the local machine will execute the equivalent of

   ```
   << /OutputDevice /Fax /FaxOptions OptionsDict >>
     setpagedevice
   file cvx exec
   ```

   resulting in a raster image fax transmission. The process terminates here, and steps 2 through 4 do not occur.

2. The **DialCallee** phone number in the *OptionsDict* dictionary is dialed and the receiver queried regarding its willingness to receive PostScript language files from the sender. The **PostScriptPassword** from the **FaxOptions** dictionary may be used in this process.

3. If the receiver is willing, the contents of the file saved internally, as well as any remainder of the file not yet read into the fax/printer, will be sent to the remote machine.

4. If the receiver is not willing, the sending machine hangs up. **RevertToRaster** from the *OptionsDict* is consulted. If it is *true* then the local machine will execute the equivalent of

   ```
   <</OutputDevice /Fax /FaxOptions OptionsDict>>
     setpagedevice
   cvx exec
   ```

   resulting in a raster image fax transmission. If **RevertToRaster** is *false*, no transmission will occur. The job will end and the message "`Callee rejected PostScript transmission`" will appear in the log.

2.3.4 The Fax Options Dictionary Keys

Both the **FaxOptions** dictionary and the argument dictionary passed to the **faxsendps** operator contain identical keys specifying the information used to direct a fax transmission. Keys in the **FaxOptions** dictionary can only be manipulated using the **setpagedevice** operator. Keys in the argument dictionary passed to the **faxsendps** operator do not affect the values in the **FaxOptions** dictionary; rather, the argument dictionary controls file transfer.

The **FaxOptions** and **faxsendps** argument dictionaries provide a number of ways of customizing fax jobs. Information in the dictionaries is placed on cover sheets and transmission reports and is recorded in logs. Procedures in

the dictionaries are used for page captions, cover sheets and transmission reports. Other values in the dictionaries control broadcast, delayed transmission and high-resolution capabilities.

Table 2.6 on page 26 lists all of the keys present in the **FaxOptions** and **faxsendps** argument dictionaries. Only these keys may be present. For each key, the allowed type of the associated value and typical default values are listed.

As described above, if there is no current page device or the new page device's **OutputDevice** is different from that of the current page device when **setpagedevice** is executed, the **FaxOptions** dictionary initially contains the keys listed below with their associated default values. Merging of the **setpagedevice**'s argument dictionary then takes place. For the **faxsendps** operator, keys not present in its *OptionsDict* argument dictionary are treated as if their product-dependent default values were given.

Table 2.6 *Fax options dictionary entries*

Key	*Type*	*Semantics*
CalleePhone	string or null	This key indicates the human-readable telephone number of the callee fax machine. The value of **CalleePhone** is used for the **Confirmation**, **CoverSheet** and **PageCaption** procedures. It differs from **DialCallee** primarily in that it omits or alters routing prefixes and suffixes. For example, the Swiss number in the example under **DialCallee** might be written here (0041-5-55-55-55732) or (CH 5-55-55-55732). If the value of **CalleePhone** is *null*, the value of **DialCallee** is used.
CallerID	string or null	The ID is defined by the ITU (1988) fax protocol and is a string of up to 20 characters which the caller uses to identify himself or herself to the callee. If it has a *null* value (the default), then the value of **ID** from the %Fax% device parameter set will be used (see Table 3.41 on page 143). If **ID** is not set, then the string returned by the system parameter **PrinterName** will be used. If this string is greater than 20 characters, then the left-most 20 characters will be used.
CallerPhone	string or null	This key indicates the human-readable telephone number of the caller fax machine.
Confirmation	procedure or null	This key prints a confirmation sheet (on the local sending print mechanism) for this job. The **Confirmation** procedure is executed when the fax job has finished and the transmission, if any, has completed. You can omit confirmation reports (by setting **Confirmation** to *null*) or customize them (by providing your own procedure). You may, for example, want to print a report only if there is an unusual printer status.

The **Confirmation** procedure must explicitly call the **showpage** operator to print the page. This allows confirmation to be sent to the host computer instead of being printed. Therefore, if an alternate definition of the confirmation report is to print upon job completion, the procedure definition must include a call to **showpage**. The details of writing these procedures are given in section 2.3.5, “CoverSheet, Confirmation and PageCaption Procedures.”

The default value depends on the value of the %Fax% parameter **DefaultConfirmOn**. If **DefaultConfirmOn** is *true* (see Table 3.41 on page 143), then the default value is

```
{/DefaultConfirmation faxopsexec}
```

which provides a simple, generic report. If **DefaultConfirmOn** is *false*, then the default value of **Confirmation** is *null*.

Note *The **faxopsexec** operator is one of the operators in the **FaxOps ProcSet**.*

Copies

array of dictionaries or null

This array of dictionaries provides a means by which the same raster or PostScript language file may be broadcast to multiple recipients. The only keys allowed in these dictionaries are the same ones allowed in the **FaxOptions** dictionaries (except that **Copies** may not be present). Each dictionary from the array is merged with the original **FaxOptions** dictionary (without the **Copies** entry) and used to direct the “mailing” of the common body of the fax to a different recipient. Where the same key is defined in both dictionaries, the value from the **Copies** element takes precedence. For more information on this broadcast facility and a discussion of some of the details and exceptions to this dictionary merging, see section 2.3.6, “Broadcast Transmission of Faxes.”

CoverNote

array of strings or null

This key passes information to the **CoverSheet** procedure. This information is intended to be a quick note on the cover sheet. It could also be used for the entire message if the fax consists of only a cover sheet.

CoverSheet

procedure or null

This PostScript language procedure generates cover sheets for the document. Fax documents are often preceded (and occasionally followed) by cover sheets. These serve some of the same purposes as envelopes for normal mail. They specify to whom the document is to be delivered, how he or she might be found, who claims to be the sender, what to do if the document is found to be incomplete, and so on. You can define **CoverSheet** as a PostScript language procedure that produces a customized page. The details for writing these procedures are given in section 2.3.5, “CoverSheet, Confirmation and PageCaption Procedures.”

Sometimes the PostScript language job will contain all of the pages that the sender wants the recipient to receive. For example, a one-page letter that contains an inside address and a return address may not require a cover sheet as well. Whenever **CoverSheet** is defined as *null*, the fax software will not generate cover pages.

The default value for this key depends on the %Fax% parameter **DefaultCoversOn** (see Table 3.41 on page 143). If this parameter is *true*, then the default is

```
{/DefaultCoverSheet faxopsexec}
```

which will generate simple cover sheets using **Sender...**, **Recipient...**, and other optional values from the **FaxOptions** dictionary. If the parameter is *false*, the default value will be *null*.

The **CoverSheet** procedure should not include a **showpage** operator. **showpage** is called automatically after the **CoverSheet** procedure is invoked.

CoverSheetOnly boolean This key indicates that it is all right to send an "empty" job. If this boolean is *false* and the PostScript language job produces no pages, then no phone call is made and nothing is sent. If this boolean is *true* and **CoverSheet** is non-*null*, then the page generated by the cover sheet procedure is sent in any case. The default value is *false*.

DialCallee string This key indicates the phone number of the fax machine to call. This string will be sent to the telephone autodialer in the fax printer. It consists of a sequence of the following characters:

P	Begin pulse dialing (initial default).
T	Begin DTMF (Dual Tone Multi-Frequency, in other words, Touch-Tone) dialing.
0-9	Send signal digit to telephone exchange.
* #	Send DTMF symbol to telephone exchange.
,	Pause — see note below.
W	Wait for dial tone.
others	Ignore.

Note *A comma internal to a dialing string will cause dialing to pause for 2 seconds before proceeding. A comma at the end of the string will cause the system to pause for up to 10 seconds (more) waiting for the call to make its way through all switches involved and a connection to be made. This can be necessary, for example, when making international calls in which the time elapsed before ringing starts is considerably longer than with domestic calls. In certain countries, limits may automatically be placed on how much of a delay is actually possible at the end of dialing. Thus, the final commas should be viewed as requests rather than absolute commands.*

For example, to dial a Swiss fax machine from a private branch exchange in the USA, you might use the string:

```
(T9,011-41-5-55-55-55732#)
```

Notice that this string contains a routing prefix (T9,011) to establish tone, international switching from the PBX (Private Branch Exchange). It also contains the suffix #, which is used by most American telephone operating companies as a signal that all digits have been sent and switching should commence immediately. This string has a maximum length of 100 characters.

ErrorCorrect boolean — Controls whether error correction should be attempted in the transmission. The receiving machine must be capable of error correction in order for the attempt to succeed. If the receiver does not have the capability, the transmission will take place without it. The default value is *true*.

FaxType integer or null — This key determines how the actual page contents are prepared and transmitted. If the value is an integer, it should be 0 or 1:

0	Use standard ITU Group 3 resolution.
1	Use fine ITU Group 3 resolution.

If the **FaxType** is 0, then the transmitted y-resolution will be approximately 100 lines per inch; if 1, the y-resolution will be twice as fine (and the transmission time will be correspondingly longer). As the ITU standards evolve, other integers may trigger other modes. If the value associated with **FaxType** is *null*, then the value of the %Fax% device parameter **DefaultResolution**, which is 0 or 1, will be used to select the resolution (see Table 3.41 on page 143).

HostJobID integer — This key allows jobs to be tagged with identifiers that can then be used to track the progress of those jobs. The value of the key is saved in the transmission logs and it is also carried with a job as it progresses through the various stages of processing and transmission. The value of this key is included in each dictionary placed on the stack by the **transmitjobsforall** (see Table 5.13 on page 198) operator from the **FaxAdminOps ProcSet** instance. The default value is 0.

MailingTime
array of integers or null — This key indicates when the fax message is to be transmitted. This is used, for example, to send documents at night when phone rates are lower. The value is an array of integers with entries as follows:

Index	*Meaning*	*Values*
0	Year	[1980-2079]
1	Month	[1-12]
2	Day	[1-31]

3	Hour	[0-23]
4	Minute	[0-59]
5	Second	[0-59]

If this key has the *null* value (the default) associated with it, then the time of job submission will be used, meaning “send immediately.” Delayed mailings are only possible if the machine has sufficient storage (disk or RAM) to save the entire job until it is to be sent.

MaxRetries integer or null — This key indicates how many additional tries after the first should be made before giving up on the transmission of a fax message. Attempts can fail, for example, because of a busy or no-answer when the call is placed. If the value is *null*, then the value used is given by the %Fax% parameter **DefaultRetryCount** (see Table 3.41 on page 143). The maximum value is 100. If an attempt is being made to send a PostScript language file and the job reverts to raster, the count of the number of failed calls, in effect, is reset to 0. This is because reverting is equivalent to submitting a new job.

nPages integer or null — This key supplies the application’s estimate of the number of pages in the job exclusive of automatically generated cover sheets. In some cases the transmitting machine has not processed the complete job before the transmission starts (and the cover sheet must be sent). In this case, the number of pages will not be known to the printer, so if **nPages** is provided, it will be used. If **nPages** is *null* (the default) then the cover sheet will list an unknown number of pages.

PageCaption
procedure or null — This PostScript language procedure generates information lines on the top of transmitted fax pages. Typically, such information will contain the sender’s name, the recipient’s name, and the current page number. The **PageCaption** procedure should not call the **showpage** operator since the procedure only makes alterations to the existing page image. The generation of captions will not occur if **PageCaption** is *null*. For more information on the interface to a **PageCaption** procedure, see section 2.3.5, “CoverSheet, Confirmation and PageCaption Procedures.”

The default value for this key depends on the %Fax% parameter, **DefaultCaptionOn** (see Table 3.41 on page 143)**.** If this parameter is *true*, then the default is the procedure

```
{/DefaultPageCaption faxopsexec}
```

If **DefaultCaptionOn** is *false*, then the default value is *null*.

PostScriptPassword
string or null — This parameter specifies a password to use in gaining permission from the callee to transmit the fax job as a PostScript language file. This entry is used only by the **faxsendps** operator. The default is *null*, which means that no password is being supplied. See section 3.6.2, “The %Fax% Device,” for a description of how this password is used.

ProcInfo dictionary or null — This dictionary may be used to supply any number of additional application-specific key-value pairs**.** These key-value pairs are used to convey variable information for application-defined cover sheets, confirmation reports and page captions.

RecipientID string or null — This key contains a string that uniquely identifies the recipient within the organization (at that fax number). It is intended to be computer readable and usable for subsequent electronic delivery of the fax message within the receiver’s organization. If this value is *null*, a string of nulls will be used. This string is placed in the Adobe Non-standard Facilities frame and becomes the **EmailDest** that the receiver logs. It also becomes the string placed in the subaddress frame when generated.

RecipientLanguage
string or null — The value is the name of the natural language to use when preparing cover sheets and page captions. If the value is *null* or a translation dictionary for the named language cannot be found, the value of the %Fax% device parameter **LocalLanguage** will be consulted. If there is a translation dictionary for the named language it will be used, otherwise the one for English will be used. See section 5.8.2 on page 199 for more information on translation dictionaries.

RecipientMailStop
string or null — This key contains information helpful for hand delivery of the fax message. For example, Mail Stop 23A, Bldg. 19.

RecipientName
string or null — This key indicates the document’s intended recipient; for example, Dr. John Doe. A *null* value for **RecipientName** causes the software to seek an alternative non-*null* value to store in the job log. The first alternative is to use the value of **RecipientOrg**. If **RecipientOrg** also has a *null* value, then the value of **CalleePhone** is used. If **CalleePhone** also has a *null* value, then the value of **DialCallee** is used. The default value is *null*.

RecipientOrg string or null — This key indicates the recipient’s company or organization name. This value is also stored in the job logs. The fall back sequence is **RecipientName** and then **DialCallee**.

RecipientPhone
string or null — This key indicates the recipient's voice telephone number. It is not the same as **DialCallee**, which is the fax phone number. The **RecipientPhone** is used for custom cover sheets. for example, that would give routing instructions to an attendant on the fax receiving end. As with **RecipientName**, if the value associated with this key is *null*, fall backs are sought to store in the log. The fall back sequence is **CalleePhone** and then **DialCallee**. The default value is *null*.

Regarding string or null — This key passes information to the **CoverSheet** procedure. This string would typically be used to add a "Subject" line to the cover page.

RetryInterval
integer or null — This key is a positive integer that specifies the number of minutes to wait before retrying to send a fax that failed. If the value is *null*, then the value is determined by the %Fax% device parameter **DefaultRetryInterval** (see Table 3.41 on page 143); the maximum value is 60 minutes.

RevertToRaster boolean — This entry is ignored by the **setpagedevice** operator but used by the **faxsendps** operator to decide what to do when the receiving machine refuses to accept a PostScript language transmission. If *true*, the PostScript language job will be imaged locally and a rasterized fax transmission will be made instead. If *false*, no transmission will occur. The default value is *true*.

SenderID string or null — This key contains a string that uniquely identifies the sender within the organization (at that fax number).

SenderMailStop
string or null — This key contains information helpful for hand delivery of a return fax message. For example, Mail Stop 43A, Bldg. 2.

SenderName string or null — This key indicates the document's sender; for example, Dr. Jane Green. A *null* value for **SenderName** causes the software to seek an alternative non-*null* value to store in the job log. The first alternative is to use the value of **SenderOrg**. If **SenderOrg** has a *null* value then **CallerID** is used. If this too is *null*, then the value of the %Fax% device parameter **ID** is used.

SenderOrg string or null — This key indicates the sender's company or organization name. If **SenderOrg** is *null*, then the value of **SenderName** is used. If **SenderName** is *null*, the value of the %Fax% device parameter **ID** is used (see Table 3.41 on page 143).

SenderPhone string or null — This key indicates the sender's voice telephone number.

TrimWhite	boolean	If **TrimWhite** is *true* when preparing a raster transmission, then white space at the top and bottom of pages will be removed before the pages are transmitted. This can result in shorter phone calls but may produce a mixture of page lengths. The default is *false*.

2.3.5 CoverSheet, Confirmation and PageCaption Procedures

The **CoverSheet**, **Confirmation** and **PageCaption** procedures in the options dictionary can be used to customize individual fax jobs. If these are not given, default procedures will be used. The default procedures are contained in a writable **ProcSet** instance named **FaxDefaultProcs**. By redefining entries in this **ProcSet** instance outside the server loop, it is possible to change the default report procedures. Initial VM is built with this **ProcSet** instance pointing at the product's built-in procedures.

The report procedure entries contained in **FaxDefaultProcs** are:

Key	*Initial Value*
DefaultCoverSheet	{/InternalDefaultCoverSheet faxopsexec}
DefaultPageCaption	{/InternalDefaultPageCaption faxopsexec}
DefaultConfirmation	{/InternalDefaultConfirmation faxopsexec}
DefaultReportJobList	{/InternalReportJobList faxopsexec}

The last item here, **DefaultReportJobList**, is the procedure behind the **reportjoblist** operator from the **FaxAdminOps ProcSet** (see section 3.6.4). Activity reports are produced using this operator.

For more information on the **FaxDefaultProcs ProcSet** instance, see section 5.8.2.

The **CoverSheet**, **Confirmation** and **PageCaption** procedures should make no change to the global state that would affect the subsequent printed appearance of the PostScript language job. These procedures are called with no parameters. However, there will be two dictionaries on the dictionary stack which provide the information required for these procedures to do their work.

The first dictionary is the **FaxOptions** dictionary. Since the **Confirmation** procedure may be run long after the fax job was processed and since the **OutputDevice** will have been set to **Printer** typically before this procedure is run, the dictionary provided to **Confirmation** is actually a copy of the **FaxOptions** dictionary from when the fax job was processed. This dictionary has the **PageCaption** and **CoverSheet** entries removed.

The second dictionary contains at least the key-value entries listed in Table 2.7.

Table 2.7 *Entries in the second dictionary used by CoverSheet, Confirmation and PageCaption*

Key	*Type*	*Semantics*
CalleeID	string	This string is defined by the ITU (1988) fax protocol as a string of up to 20 characters with which the callee can identify himself to the caller. It is transmitted between the two stations when they first handshake with each other. It is not present in the dictionaries provided to the **CoverSheet** and **PageCaption** procedures. For multiple call transmissions, the value determined on the first phone call is used.
CallLength	integer	This key provides the number of seconds that the transmission session lasted. It is not present in the dictionaries provided to the **CoverSheet** and **PageCaption** procedures.
CoverType	integer	This key is only meaningful to the **CoverSheet** procedures. It has a value indicating the type of cover needed. 0 Front cover 1 Back cover
CurrentPageNo	integer	This is the number of the page currently being prepared. This value can be used for generating page captions. Cover sheets are not included in this running count.
ErrorArray	array of strings	This array describes particular error conditions. It is indexed by **ErrorIndex**.
ErrorIndex	integer	This integer can be used to retrieve a string from **ErrorArray** describing the status of this job.
IncludesFinalPage	boolean	This key indicates whether this session is the last and final transmission session for this job. If *true*, then this session includes the last page of the overall job; if *false*, it does not. For more information on transmission sessions, see **InitialPage** and **LimitPage**.
InitialPage and **LimitPage**	integer	This key determines which pages of an overall job have been or will be sent in this transmission. The job may get broken into more than one distinct transmission session due to errors on the telephone line, overruns or underruns, and so on. If cover pages are being generated, then each distinct transmission will have its own cover sheet. The two items here can be used by the cover page procedure to figure out how many pages are in a particular transmission and where they fall with respect to other transmissions (if any). **InitialPage** is the page number of the first page of the session less one. (That is, it is 0 if this transmission includes the first page of the job.) **LimitPage** is the total number of pages in this session and all preceding sessions of this job excluding all cover sheets.

NumberOfCalls integer This key specifies the number of separate telephone calls used to make the transmission. It is not meaningful in the dictionaries provided to the **CoverSheet** and **PageCaption** procedures.

PagesSent integer This key specifies a count of the number of pages sent to the destination fax machine. It is not present in the dictionaries provided to the **CoverSheet** and **PageCaption** procedures.

SendPostScript boolean This key indicates whether the job is a PostScript language file transmission. A value of *true* indicates that it is; *false* indicates that it is not.

TimeSent array of integers This array contains date and time information.

Index	*Meaning*	*Values*
0	Year	[1980-2079]
1	Month	[1-12]
2	Day	[1-31]
3	Hour	[0-23]
4	Minute	[0-59]
5	Second	[0-59]

For confirmation reports, this is the time of the first call (of possibly many) made for the job. For cover sheets and page captions, this is the time the PostScript language job was submitted and processed.

2.3.6 Broadcast Transmission of Faxes

The purpose of providing a broadcast capability for fax is to optimize throughput. For a raster fax, the common body is rasterized and compressed only once. For a PostScript language file transmission, the out-bound file is assembled only once. Then, in either case, the previously prepared material is transmitted multiple times—one copy to each different recipient specified in the **Copies** array. Thus, one requirement for a broadcast job to succeed is that there must be sufficient storage (RAM or disk) to hold all outgoing files that form the fax. Broadcast transmissions are not broken into multiple calls.

As described earlier, the entries in the **Copies** array are individually merged with the original **FaxOptions** dictionary or **faxsendps** argument dictionary to produce a sequence of new dictionaries which are used to direct the mailings. Where the same key is defined in both the original dictionary and the **Copies** element, the value from the **Copies** element takes precedence (except as noted in the following discussion).

When raster is being sent, most of the **FaxOptions** entries provide mailing information and thereby determine how the fax is sent; however, a few of the entries actually affect what is to be sent. For those entries which affect the

common pages to be sent, the values in the original **FaxOptions** dictionary prevail and the ones in the **Copies** dictionaries are ignored. The entries in this category are: **FaxType**, **TrimWhite** and **PageCaption**.

The **PageCaption** procedure designated by the original **FaxOptions** will be used to place captions on the common pages (or no captions, if that is desired). Two dictionaries are placed on the dictionary stack before calling the **PageCaption** procedure. One of these is a **FaxOptions** dictionary. If broadcast pages are being prepared, the original **FaxOptions** will be used. This means that accesses to items such as **RecipientName** and **RecipientPhone** will retrieve the values from this dictionary and not from any of the merged dictionaries. Thus, giving these keys values like (Distribution) or (Mailing List), makes sense if the **PageCaption** procedure will be using them.

If a raster is to be broadcast, individual cover sheets will be produced for each recipient, as determined by the **CoverSheet** procedures in the merged **FaxOptions** dictionaries. These cover sheets are imaged separately from the common pages and may contain recipient-specific information derived from the merged **FaxOptions** dictionaries.

When a PostScript language file is being broadcast and cover sheets are asked for, the first part of the file sent to each recipient contains code to reproduce a dictionary. The dictionaries are used by the receivers to create customized cover sheets specific to each receiver. The dictionaries which are coded up are derived from the merged dictionaries, not just the top level **FaxOptions** dictionary.

Different **Copies** entries may have different values for the **RevertToRaster** key, just as they may each have a different **PostScriptPassword**. When a job asks to broadcast a PostScript language file, that is all that is initially prepared. Transmission starts and proceeds until some recipient requires raster. At this point, rasterization and compression will begin. The phoning and transmitting of a PostScript language file continues at the same time. If other recipients are found which demand raster, they are recorded so that when the common rasterization is performed, a new phone call can be placed. Cover sheets are imaged separately, as needed.

The value of **nPages** in different **Copies** dictionaries may vary, although this would seem to be illogical. The value of **nPages** is irrelevant in the context of broadcast anyway since the entire job must be rasterized before any transmissions take place. It is only used if transmission must start before imaging is done.

CoverSheetOnly is another key whose values may vary. It may have different values in different dictionaries. However, this could mean that as a result of the broadcast, some recipients will get one page (a cover sheet) and others will not even be called.

2.3.7 Some Sample Fax Jobs

This section presents several example PostScript fax jobs. These examples show some of the major features of the job interface.

A Simple Example Sending a Raster File

This snippet of PostScript language code produces a cover page (if the value of the %Fax% parameter **DefaultCoversOn** is *true*; see Table 3.41 on page 143) and the document page on the fax machine answering the phone at 415-555-3710.

```
2 dict dup
begin
  /OutputDevice /Fax def
  /FaxOptions 10 dict dup begin
    /DialCallee (T9,1-415-555-3710) def
    /RecipientName (Joe Smith) def
    /FaxType 0 def
    end
  def
end
setpagedevice

% now comes the real document
/Bookman-Light findfont 20 scalefont setfont
100 400 moveto (Hello, world!) show
showpage
```

A transmission report will also be produced on the local printer.

An Example with User-Defined Procedures

This is an extension of the previous example, and shows the use of custom **Cover sheet**, **PageCaption** and **Confirmation** procedures.

```
2 dict dup
begin
  /OutputDevice /Fax def
  /FaxOptions 10 dict dup begin
    /DialCallee (T9,1-415-555-3710) def
    /RecipientName (Joe Smith) def

  % A simple coversheet
    /CoverSheet {
      /Times-Roman findfont
      40 scalefont setfont
      200 400 moveto
      (FAX Cover) show
    } def
```

```
  % A different page caption
    /PageCaption {
      /Times-Roman findfont
      8 scalefont setfont

      % put caption at very top of page
      300 755 moveto

      (Communicate by Fax!) show
    } def

  % Confirmation report - prints locally
    /Confirmation {
      /Times-Roman findfont
      40 scalefont setfont
      200 400 moveto
      (Fax Transmission Report) show
      200 200 moveto
      (Pages: ) show

      % access one of the dictionaries
      PagesSent 5 string cvs show

      showpage
    } def
    end
  def
end
setpagedevice

% now comes the real document
/Bookman-Light findfont 20 scalefont setfont
100 400 moveto (Hello, world!) show
showpage
```

A Simple Example Sending a PostScript Language File

This example sends the same fax message as the first example, but does it by sending the PostScript language file instead of a rasterized image.

```
currentfile
10 dict dup begin
  /DialCallee (T9,1-415-555-3710) def
  /RecipientName (Joe Smith) def
  /FaxType 0 def
  /RevertToRaster true def
  end

/FaxOps /ProcSet findresource /faxsendps get exec
```

```
% now comes the real document
/Bookman-Light findfont 20 scalefont setfont
100 400 moveto (Hello, world!) show
showpage
```

Note *If the fax machine being called is not willing to accept a PostScript language file (or if it requests a password, which the above code has not provided), the fax printer will hang up, image the page locally and then transmit the rasterized page with a second phone call.*

2.4 Envelope Orientation in User Space

This section describes how default user space is oriented relative to the flap on an envelope. This discussion assumes that the **Install** procedure does not alter the default transformation matrix.

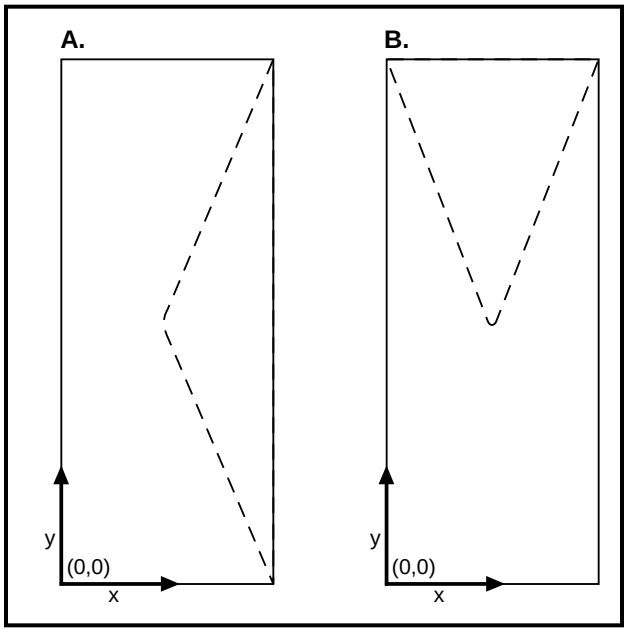
If the **PageSize** value is portrait ([width height] with width < height), then default user space is set up so that the origin is on the opposite edge of the envelope from the flap and in the diagonally opposite corner from the return address (on a U.S. business envelope). The default user space is set up this way regardless of how envelopes are fed into the printer on a particular product.

Figure 2.1 illustrates two envelopes: one with its flap along the long edge of the envelope, and one with its flap along the short edge of the envelope. The dashed line indicates that the flap is on the side of the envelope facing down.

If the flap is along the long edge of the envelope, then default user space for a portrait **PageSize** is set up as in panel A of Figure 2.1.

If the flap is along the short edge of the envelope, then the default user space for a portrait **PageSize** is set up as in panel B of Figure 2.1.

Figure 2.1 *Default user space orientation for a portrait envelope PageSize*



For landscape **PageSize** values ([width height] with width > height), the orientation of default user space is defined relative to the orientation for portrait **PageSize** values. This relationship is described in Table 4.10 in section 4.11 of the *PostScript Language Reference Manual, Second Edition*.

2.5 Errors Generated by Page Device Parameters

In addition to a **configurationerror**, the **setpagedevice** operator can generate a **typecheck**, **rangecheck**, **undefined**, **limitcheck** or **invalidaccess** error under certain conditions.

If a feature is unknown for a product, then policy is invoked for that feature, without checking the type of the value. Therefore, the only error that can be generated for unknown features is a **configurationerror**, and only if the policy specifies that this is to be done. For most products, the default policy for unknown features is to ignore them.

2.5.1 typecheck Errors

A **typecheck** error is generated if:

- The type of the value for a feature is not one of the acceptable types for that feature, or a component value within a compound value is not the correct type. Each of the following examples would generate a **typecheck** error:

```
<< /BeginPage 4 >> setpagedevice
```
This example generates a **typecheck** error.

```
<< /Margins [0 true] >> setpagedevice
```
This example generates a **typecheck** error.

```
<< /InputAttributes << 0 23 >>  >> setpagedevice
```
This example generates a **typecheck** error.

- A literal array is given for a value that should be a procedure. However, an executable array is acceptable wherever an array value is expected. Packed arrays are always acceptable wherever an array is acceptable. The first two examples below would generate a **typecheck** error; the third would not:

  ```
  << /Install [2 3 4] >> setpagedevice
  ```
  This example generates a **typecheck** error.

  ```
  << /Policies <</PolicyReport [5 6 7] >>  >> setpagedevice
  ```
  This example generates a **typecheck** error.

  ```
  << /PageSize {612 792} >> setpagedevice
  ```
  This example is correct.

- The operand to **setpagedevice** is not a dictionary. The following example would generate a **typecheck** error:

  ```
  true setpagedevice
  ```
  This example generates a **typecheck** error.

2.5.2 rangecheck Errors

A **rangecheck** error is generated if:

- An array value of the wrong length is given, either as the value for a feature, or as a component of a value within a compound value. Each of the following examples would generate a **rangecheck** error:

  ```
  << /HWResolution [300] >> setpagedevice
  ```
  This example generates a **rangecheck** error.

  ```
  << /InputAttributes << 0 << /PageSize [600 700 800] >>  >>  >>
  setpagedevice
  ```
  This example generates a **rangecheck** error.

- A value of the right type, but beyond the acceptable range of values, is given either as the value for a feature, or as a component of a value within a compound value. Each of the following examples would generate a **rangecheck** error:

```
<< /PreRenderingEnhanceDetails << /Type –1 >>  >>
setpagedevice
```
This example generates a **rangecheck** error.

```
<< /Jog 10 >> setpagedevice
```
This example generates a **rangecheck** error if **Jog** is known.

2.5.3 undefined Errors

An **undefined** error is generated if:

- The **Type** key is not specified in a **Details** dictionary. For example:

  ```
  << /Fold 4 /FoldDetails <</FoldType (ZFold) >> >>
  setpagedevice
  ```
  This example generates an **undefined** error. The **Type** key is mandatory in the **FoldDetails** dictionary and it is missing in this example.

2.5.4 invalidaccess Errors

An **invalidaccess** error is generated if:

- A string, array, or dictionary value is given whose access is more restrictive than *read-only*, either as the value for a feature or as a component value within a compound value. An exception is that for values that are procedures, the value can be *execute-only*. The first two examples below would generate **invalidaccess** errors; the third would not:

  ```
  << /MediaColor (blue) noaccess >> setpagedevice
  ```
  This example generates an **invalidaccess** error.

  ```
  << /PageSize {612 792} executeonly >> setpagedevice
  ```
  This example generates an **invalidaccess** error.

  ```
  << /BeginPage {pop} executeonly >> setpagedevice
  ```
  This example is correct.

- The operand to **setpagedevice** is a dictionary whose access is more restrictive than *read-only*. The following example would generate an **invalidaccess** error:

  ```
  << /PageSize [612 792] >> noaccess setpagedevice
  ```
  This example generates an **invalidaccess** error.

2.5.5 limitcheck Errors

A **limitcheck** error is generated if there is insufficient storage during the invocation of **setpagedevice**.

CHAPTER 3

Interpreter Parameters

The various interpreter parameters control the operation and behavior of the PostScript interpreter. Many of them have to do with allocation of memory and other resources for specific purposes. For example, there are parameters to control the maximum amount of memory used for VM, font cache, and halftone screens. Some input/output devices have parameters that control the behavior of each device individually.

A printer is initially configured with interpreter parameter values that are appropriate for most applications. However, a PostScript language program can alter the interpreter parameters to favor a certain type of functionality or to adapt the product to special requirements. There are three classes of interpreter parameters: user, system and device parameters.

For each class there is a PostScript operator to read the parameter values and an operator to set the parameter values. The resulting six operators are **currentuserparams**, **setuserparams**, **currentsystemparams**, **setsystemparams**, **currentdevparams**, and **setdevparams**.

Refer to the *PostScript Language Reference Manual, Second Edition*, Chapter 8 for descriptions of these operators and to Appendix C in the same manual for further information about interpreter parameters.

3.1 Two Kinds of Unencapsulated Jobs

An unencapsulated job is entered by executing the Level 2 operator **startjob** or the Level 1 operator **exitserver**. These operators require a password to be presented. The password must be equal to the value of either the **StartJobPassword** or the **SystemParamsPassword** system parameter. If the password is equal to the value of **StartJobPassword**, an ordinary unencapsulated job is started (see section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*). If the password is equal to the value of **SystemParamsPassword**, a system administrator job is started. (If the **SystemParamsPassword** is a zero-length string or has never been set, every unencapsulated job is a system administrator job.)

3.2 Passwords for System and Device Parameters

The system parameters **StartJobPassword** and **SystemParamsPassword** are explained in section C.3.1, “Passwords,” of the *PostScript Language Reference Manual, Second Edition*. Section C.4 makes the statement “**setdevparams** is very similar to **setsystemparams**; the same restrictions apply.” This needs to be clarified a little. When setting device parameters, most but not all will require a password equal to **SystemParamsPassword**. Also, there is one system parameter that does not require a password. The exceptions to the rules are as follows:

- The **FactoryDefaults** system parameter does not require a password if **FactoryDefaults** is the only entry in the dictionary passed to **setsystemparams**. If the only other key in the dictionary is the password, it is ignored. This is necessary so that if the **SystemParamsPassword** has been forgotten, there will still be a way to reset it (see **FactoryDefaults** described in Table 3.2 on page 47).

- The device parameters **Interpreter** and **Protocol** found in device sets of type /Communications do not require a password if one or both are the only entries in the dictionary passed to **setdevparams**. If the only additional key in the dictionary is the password, it is ignored (see **Interpreter** described in Table 3.4 on page 66 and **Protocol** described in Table 3.5 on page 71).

3.3 User Parameters

Any PostScript language program can set user parameters during job execution; no password is required. The initial value of user parameters when the printer is turned on for the first time is product-dependent.

Unless otherwise specified, all user parameters are subject to **save** and **restore**. (At this time, **JobTimeout** is the only parameter that does not obey **save** and **restore**.) This means that if an unencapsulated job changes user parameters, these new values are the initial values for subsequent encapsulated jobs. There are exceptions to this generalization. For a system parameter whose name is the same as a user parameter, the value of the system parameter is used to initialize the corresponding user parameter at the beginning of each job. In any case, changes made to any user parameter by an encapsulated job have no effect on the initial value of user parameters for subsequent jobs.

User parameters are maintained on a per context basis in environments that support multiple contexts.

The following user parameters are described in Table C.1, Appendix C of the *PostScript Language Reference Manual, Second Edition*. The description of these parameters is unchanged.

MaxDictStack	**MaxExecStack**	**MaxFontItem**
MaxFormItem	**MaxLocalVM**	**MaxOpStack**
MaxPatternItem	**MaxScreenItem**	**MaxUPathItem**
MinFontCompress	**VMReclaim**	**VMThreshold**

Each user parameter is identified by a key, which is always a name object. The value of the parameter is usually an integer.

Table 3.1 describes user parameters that have been defined or amended since publication of the *PostScript Language Reference Manual, Second Edition.*

Note *In this table, ‡ means that this key is typically present in all job server (that is, printer) implementations.*

Table 3.1 *User parameters*

Key	*Type*	*Semantics*
AccurateScreens	boolean	This parameter controls whether the accurate screen algorithm is used during subsequent executions of the **setscreen** and **setcolorscreen** operators. This parameter has no effect on screens established by **sethalftone**. See section 6.4.4 of the *PostScript Language Reference Manual, Second Edition*, for a description of accurate screening for the **sethalftone** operator. **Legal values:** *true*, *false* **Errors:** None.
JobName ‡	string	This parameter establishes *string* as the name of the current job. If defined as a non-zero length string, status responses generated during the remainder of the current job will include a job field that reports the text of this string. The characters should be within the ASCII printable range, because this information is transmitted across arbitrary communications channels and is intended for display to users. **Legal values:** Any sequence of byte values up to an implementation-dependent maximum length. However, it should not contain the characters ‘;’ or ‘]’ because they would disrupt the syntax of status messages. If the maximum length is exceeded, the string is truncated. **Errors:** **limitcheck**, **typecheck**

JobTimeout integer Setting **JobTimeout** to a positive value establishes this value as the current job time-out, the number of seconds a job is allowed to execute before it is aborted and a PostScript language **timeout** error is generated. The current value is decremented during the job, and reading it returns the number of seconds remaining before the job time-out will occur. Time spent waiting for communications and correcting device-error conditions is not considered as part of the job execution time. Setting this parameter to 0 disables job time-out altogether.

JobTimeout is not subject to **save** and **restore**. It is initialized to the value of the **JobTimeout** system parameter at the beginning of each job.

Legal values: Any non-negative integer.

Errors: **typecheck**

WaitTimeout ‡ integer This parameter indicates the current wait time-out, which is the number of seconds the interpreter waits to receive additional characters from the host before it aborts the current job by executing a PostScript language **timeout** error. A value of 0 indicates an infinite time-out. This parameter is initialized to the value of the **WaitTimeout** system parameter at the beginning of each job.

Legal values: Any non-negative integer.

Errors: **typecheck**

3.4 System Parameters

In general, setting system parameters requires a password. System parameter values persist across jobs. (Depending upon the product, some system parameters are stored in non-volatile memory and are persistent across restarts of the interpreter.)

System parameters are global to the PostScript language environment and, in particular, are not maintained on a per context basis in the environments that support multiple contexts. The initial value of system parameters when the device is turned on for the first time and which parameters are stored in non-volatile memory depends on the product implementation.

Some system parameters are *read-only.* That is, they are returned by **currentsystemparams**, but any attempt to change one using **setsystemparams** has no effect. Other parameters are *write-only.* They can be set by **setsystemparams**, but are not returned by **currentsystemparams**.

Each system parameter is identified by a key, which is always a name object. The following system parameters are described in the *PostScript Language Reference Manual, Second Edition*. The description of these parameters is unchanged.

Note *In the following lists, as well as in Table 3.2, ‡ means that this key is typically present in all job server (that is, printer) implementations.*

ByteOrder	**CurDisplayList**	**CurFontCache**
CurFormCache	**CurOutlineCache**	**CurPatternCache**
CurScreenStorage	**CurUPathCache**	**DoPrintErrors**
MaxDisplayList	**MaxFontCache**	**MaxFormCache**
MaxOutlineCache	**MaxPatternCache**	**MaxScreenStorage**
MaxUPathCache	**RealFormat**	

Table 3.2 describes system parameters that have been defined or amended since the publication of the *PostScript Language Reference Manual, Second Edition*.

Table 3.2 *System parameters*

Key	*Type*	*Semantics*
BuildTime	integer	(*Read-only*) A time stamp identifying a specific build of the PostScript interpreter. The values returned by **BuildTime** on two different products need not be comparable, and in general, **BuildTime** should only be interpreted in conjunction with the manufacturer's product documentation. **Legal values:** Any integer. **Errors:** None.
CompressImageSource	boolean	When *true*, a compression filter will be applied to the image source data where such compression benefits the product in terms of memory use or performance. **Legal values:** *true, false* **Errors:** **typecheck**
CurBufferType	name	(*Read-only*) This parameter (typically found on imagesetters) indicates information about how the raster memory is used. The choices are /Band and /Hybrid. /Band indicates that the system will render to bands (groups of scan lines), regardless of the amount of RAM available. When /Hybrid is returned,

if **MaxRasterMemory** is large enough to contain a full-page frame buffer, the interpreter will render into the full-page buffer. Otherwise, it will render to bands.

Legal values: /Band, /Hybrid

Errors: None.

CurInputDevice string (*Read-only*) This parameter indicates the name of the communications device corresponding to the current input file for the currently executing PostScript language program. The string that is returned corresponds to the communications device parameter set name whose values are normally stored in RAM; for example, (%Serial%). For more information on communications devices, see section 3.5.2, “Device Parameters Associated with Communications.”

Legal values: A string containing a communications device name.

Errors: None.

CurOutputDevice string (*Read-only*) This parameter indicates the name of the communications device corresponding to the current output file for the currently executing PostScript language program. The string that is returned corresponds to the communications device parameter set whose values are normally stored in RAM; for example, (%Serial%). For more information on communications devices, see section 3.5.2, “Device Parameters Associated with Communications.”

Legal values: A string containing a communications device name.

Errors: None.

CurSourceList ‡ integer (*Read-only*) This parameter indicates the number of bytes currently occupied by source lists. A source list holds the internal data representation for sampled image source data and uncached character pixel arrays.

Legal values: Any non-negative integer.

Errors: None.

CurStoredFontCache integer (*Read-only*) This parameter indicates the number of bytes that the storage device font cache currently occupies.

Legal values: Any positive integer.

Errors: None.

CurStoredScreenCache

integer (*Read-only*) This parameter indicates the number of bytes currently used for screen files on the storage device that includes the currently active screens.

Legal values: Any positive integer.

Errors: None.

DoPrintErrors boolean This parameter indicates whether the built-in error handler for the product is enabled. All PostScript printers have an error handler to catch errors that are generated by programs. See section 3.10.2 of the *PostScript Language Reference Manual, Second Edition* for more information. Some printers also have a built-in error printer that can be enabled to print the error and stack contents on the current partial page, much like the one described in Appendix A of *PostScript Language Program Design*. The system parameter **DoPrintErrors** determines whether this error printing is enabled. This system parameter is present only in printers that have such a built-in error printer. Any printer that supports LaserJet 4 emulation will have one and can be controlled either from this system parameter or from the PJL commands:

```
@PJL [SET | DEFAULT] LPARM: POSTSCRIPT PRTPSERRS = [ON | OFF]
```

Legal values: *true*, *false*

Errors: **typecheck**

DoStartPage boolean This parameter indicates whether the start page should print during system initialization. The start page prints if the value of **DoStartPage** is *true* during system initialization.

Legal values: *true*, *false*

Errors: **typecheck**

EnvironmentSave boolean In systems with multiple page description languages (PDLs), this parameter controls whether or not the system can reclaim memory belonging to dormant PDLs when the system runs out of memory. If **EnvironmentSave** is *true*, all permanent objects belonging to all PDLs persist across PDL switches. If **EnvironmentSave** is *false*, all memory belonging to dormant PDLs can potentially be reclaimed when the system runs out of memory.

Setting **EnvironmentSave** to *true* at low memory configurations could make the system essentially unusable. In low memory configurations, therefore, this parameter should be a constant value of *false*.

When the memory installed in the system is above the product-defined limit, this parameter can be set by the user. The default value is *true*.

Whenever the value of **EnvironmentSave** is changed, the new value is effective immediately (the system does not have to be rebooted).

Legal values: *true*, *false*

Errors: **typecheck**

FactoryDefaults boolean

This parameter is usually *false*. Setting it to *true* and immediately turning off the printer causes all non-volatile parameters to revert to factory default values at the next power-on. The job that sets **FactoryDefaults** to *true* must be the last job executed before power-off; otherwise, the request is ignored. This required physical interaction reduces the chance of malicious jobs resetting the device to factory defaults.

A password is not required in the dictionary passed to **setsystemparams** if **FactoryDefaults** is the only entry in the dictionary. This allows the factory defaults to be reestablished even though the system parameters password might have become corrupted.

Note *The passwords are among those reset by this operation.*

The exact collection of parameters reset to factory defaults by this action is product-dependent. In most products, **PageCount** is not reset.

Legal values: *true*, *false*

Errors: **typecheck**

FatalErrorAddress ‡ integer

A fatal system software error causes a PostScript output device to stop execution and, in most products, to restart the PostScript interpreter. Before execution is stopped, the address at which the error occurs is stored in the parameter **FatalErrorAddress** and also is transmitted to the host over the communications channel. A non-zero value of this parameter indicates that a fatal system error has occurred earlier. On some products, if this value is non-zero during system initialization, the address is printed on the start page or possibly on a separate page.

Legal values: Any integer.

Errors: None.

FontResourceDir ‡ string

This parameter controls the location of external fonts. Fonts are resources in PostScript language Level 2. The **Font** category implementation concatenates the **FontResourceDir** and the font name to get the external location of the font. For example, if the **FontResourceDir** were (Resource/Font/), then the Times-Roman resource of the **Font** category would be in (Resource/Font/Times-Roman).

This parameter is provided separately from the **GenericResourceDir** system parameter to allow backward compatibility with applications that expect fonts to be located under (fonts/). In such a case, **FontResourceDir** should be set to (fonts/).

Note *Applications and users should access external fonts only through the resource operators or* ***findfont*** *or, if necessary to access them as files, through* ***ResourceFileName****. (See the PostScript Language Reference Manual, Second Edition, section 3.9, "Named Resources.") The above parameter should be used only to control the location of external fonts by the resource management mechanism.*

Legal values: Any string with non-null characters.

Errors: **limitcheck**, **typecheck**

GenericResourceDir ‡ and **GenericResourcePathSep** ‡

strings These parameters control the location of external resources for the **Generic** category and all categories based upon it (currently **Category**, **Encoding**, **Form**, **Pattern**, **ProcSet**, **ColorSpace**, **Halftone** and **ColorRendering**). The **Generic** category implementation concatenates the **GenericResourceDir**, the category name, the **GenericResourcePathSep** and the resource name to get the external location of the resource. For example, if **GenericResourceDir** and **GenericResourcePathSep** were (Resource/) and (/), respectively, then the AdobeLogo resource of the **Pattern** category would be in Resource/Pattern/AdobeLogo.

The **GenericResourceDir** should be an absolute path, that is, a path beginning at the root of the storage device. It must contain any trailing path separator. It should include a storage device (for example, %os%) if only a single device is to be considered, or it should omit the device if all searchable devices are to be considered. If there is a device specifically for generically managed resources (for example, %GenericResource%) that may access resources through a network server or along a search path, then **GenericResourceDir** should be set to that device. Resource files are expected to be in subdirectories with names the same as category names. The resource file name should be the same as the name of the resource it defines. In the above example, the file named Resource/Pattern/AdobeLogo should contain a PostScript language program which, when run, will define the AdobeLogo instance in the Pattern resource category.

Note *Applications and users should access external resources only through the resource operators or, if necessary to access them as files, through* ***ResourceFileName****. (See the PostScript Language Reference Manual, Second Edition, section 3.9, "Named Resources.") The above parameters should be used only to control the location of external resources by the resource management mechanism.*

For products with no external resources (and presumably, no file systems), **GenericResourceDir** should be set to (%null). This mechanism can also be used by site administrators to temporarily disable access to external resources.

Legal values: Any string with non-null characters.

Errors: **limitcheck**, **typecheck**

InstalledRam integer (*Read-only*) This parameter indicates, in bytes, the total amount of installed RAM in the system (**InstalledRam** should not be confused with **RamSize**, which is the amount of RAM available to the page description language.)

Legal values: Any positive integer.

Errors: None.

JobTimeout integer This parameter indicates the value in seconds to which the user parameter **JobTimeout** is initialized at the beginning of each job. Trying to set the system parameter **JobTimeout** to a negative value is ignored and the previous setting of **JobTimeout** is used. A value of 0 indicates that the time-out is infinite. Trying to set a number between 1 and 14 will result in 15 being set (in other words, 15 is the minimum value). The reason for the minimum value of 15 is that if small values were allowed, this might prevent a subsequent job from setting **JobTimeout** to another value successfully.

Legal values: 0 or any integer greater than or equal to 15.

Errors: **typecheck**

LicenseID string This parameter contains the Adobe-assigned license identifier. Its value is unique to each product.

Legal values: Any string of non-null characters.

Errors: **limitcheck**, **typecheck**

MaxHWRenderingBuffer
integer This parameter indicates the amount of memory, in bytes, to reserve for use by hardware rendering devices, such as PixelBurst™ or ColorBurst™, to store display list data. The memory is permanently allocated during system initialization. If the value being set is outside of the legal range, **MaxHWRenderingBuffer** is set to the nearest acceptable value. The

minimum value meets the requirements of the rendering device and the maximum value is an amount that will not jeopardize the execution of a PostScript language job.

Legal values: Product-dependent. Any positive integer, typically 8192 or greater.

Errors: None**.**

MaxImageBuffer ‡ integer This parameter indicates the maximum number of bytes that can be utilized for a single image buffer. An image buffer holds an internal data representation for sampled image source data. The parameter may be rounded by the interpreter if a requested value is out of range.

Legal values: Any integer.

Errors: **typecheck**

MaxPermanentVM integer This parameter defines the upper limit, in bytes, of the amount of permanent VM that can be downloaded when **EnvironmentSave** is *true*. The upper limit of permanent VM is the VM at save level zero defined by unencapsulated PostScript language jobs. This limit is not enforced if **EnvironmentSave** is *false*.

If **EnvironmentSave** is *true*, any attempt to download more permanent VM than defined by **MaxPermanentVM** will generate a **VMerror**. If a user attempts to set **MaxPermanentVM** to less than the current permanent VM plus a small threshold value, **MaxPermanentVM** will default to the smallest allowable value.

Whenever the value of **EnvironmentSave** is reset to *true*, if **MaxPermanentVM** is set to a value lower than the smallest allowable value, **MaxPermanentVM** will default to the minimum allowable value.

. The **MaxPermanentVM** parameter is present in the system parameter set only if **EnvironmentSave** is defined.

Legal values: Any integer.

Errors: None.

MaxRasterMemory integer This parameter indicates the largest amount of memory, in bytes, that may be allocated to the frame buffer. This parameter may be used to limit the amount of raster memory; unused raster memory is available for use as VM. Thus, **MaxRasterMemory** allows the user to trade-off raster memory allocation (which will allow larger page sizes and higher resolutions) against VM (which will allow more downloaded fonts and the production of more

complex pages). **MaxRasterMemory** is consulted only during system initialization; any changes to the value of the parameter will not take effect until then.

Legal values: Product-dependent.

Errors: **typecheck**

MaxSourceList ‡ integer This parameter indicates the maximum number of bytes that can be utilized for source lists. A source list holds internal data representation for sampled image source data and uncached character pixel arrays. This parameter may be rounded by the interpreter if a requested value is out of range.

Legal values: Any positive integer.

Errors: **typecheck**

MaxStoredFontCache

integer This parameter defines the maximum number of bytes that the storage device font cache can occupy on the chosen storage device (such as the disk). Setting **MaxStoredFontCache** to 0 has the effect of turning off stored caching. Setting **MaxStoredFontCache** to -1 (or to a value too large), sets the number of bytes that the font cache can occupy to the logical size of the storage device. If the logical size of the storage device is not known, an implementation-dependent value is used.

Legal values: -1, 0, or any positive integer.

Errors: **typecheck**

MaxStoredScreenCache

integer This parameter defines the maximum number of bytes that the storage device screen cache can occupy on the chosen storage device. Setting **MaxStoredScreenCache** to 0 turns off stored caching. Setting **MaxStoredScreenCache** to a negative value (or to a positive value too large) sets the number of bytes that the screen cache can occupy to the logical size of the storage device. If the logical size of the storage device is not known, an implementation-dependent value is used.

Legal values: Any integer.

Errors: **typecheck**

MinBandBuffers integer This parameter (typically found on imagesetters) is used to specify the minimum number of band buffers (groups of scan lines) to be allocated from memory set aside as raster memory. The default value depends on the product and the amount of memory installed. Typically on imagesetters, the default is 2 for configurations with 32 megabytes of memory or less and 3 otherwise.

Legal values: Any positive integer.

Errors: None.

PageCount integer (*Read-only*) The **PageCount** parameter indicates the number of pages that have successfully been processed since manufacture. The **PageCount** parameter is incremented when the interpreter finishes executing each page. The page count is incremented at these times by the value of the current copy count. If one or more pages are not actually printed for any reason, including manual feed time-out and job abort, **PageCount** is not decreased accordingly. In most products, **PageCount** is not reset at a user request to return to factory defaults. However, **PageCount** may be reset if the non-volatile memory in which it is stored has been corrupted.

Note *In releases prior to PostScript Language version 2014, the **PageCount** parameter is incremented when a page completes printing, rather than during execution of the **showpage** or **copypage** operators.*

Legal values: Any non-negative integer.

Errors: None.

PrinterName string This parameter establishes *string* as the current name of the device. If the device is on a network, this name might be used by the system as part of a name identifier for the device considered as a node on the network. **PrinterName** is usually printed on the start page and so it should consist of printable characters, although this is not required. Setting this parameter to a zero-length string causes **PrinterName** to be set to the value of the **product** string in **systemdict**.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **limitcheck**, **typecheck**

RamSize integer (*Read-only*) This parameter indicates, in bytes, the amount of installed RAM available to the PDL. In some cases, this value might be less than the total amount of installed RAM in the product. For example, the system diagnostics might have determined that certain banks of RAM are defective and would consider them unavailable.

Legal values: Any positive integer.

Errors: None.

Revision integer (*Read-only*) This parameter designates the current revision level of the product in which the PostScript interpreter is running. Each product has its own numbering system for revisions, independent of those of any other product. The value is identical to the value of the integer **revision** in **systemdict**.

Legal values: Any integer.

Errors: None.

StartJobPassword ‡ string If a program starts an unencapsulated job using **startjob** or **exitserver**, and if the password it presents to that operator is the value of **StartJobPassword**, then the subsequent unencapsulated job will need to present a password equal to the **SystemParamsPassword** each time **setsystemparams**, **setdevparams** or other system administrator operations are invoked.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **limitcheck**, **typecheck**

StartupMode integer This parameter controls whether the system start file (Sys/Start) or some other start-up procedure should be executed during system initialization. The Sys/Start file executes if the value of **StartupMode** is 1 during system initialization. If the **StartupMode** value is 0, no special start-up procedures are run during system initialization. Other values of **StartupMode** can occur in specific products and result in product-dependent start-up procedure execution.

Legal values: Product-dependent, but restricted to values between 0 and 255.

Errors: **typecheck**

SystemParamsPassword ‡ string If a program starts an unencapsulated job using **startjob** or **exitserver**, and if the password it presents to that operator is the value of **SystemParamsPassword**, then the subsequent unencapsulated job is permitted to invoke **setsystemparams**, **setdevparams**, or other system

administrator operations without presenting a password each time. This extends to Level 1 compatibility operators that change system parameters but provide no means to present a password.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **limitcheck**, **typecheck**

ValidNV boolean (*Read-only*) This parameter indicates whether non-volatile memory is currently used to store persistent parameters. During system initialization, if non-volatile memory is corrupt, factory defaults are reestablished. If further testing indicates that non-volatile memory is defective, it will not be used, and **ValidNV** is *false*; otherwise, **ValidNV** is *true*. In many products, if non-volatile memory is defective, it is emulated in RAM. The operating behavior is the same, except that persistent parameter values are lost when the printer is powered off or restarted and factory defaults are used at power-on.

Legal values: *true*, *false*

Errors: None.

WaitTimeout ‡ integer This parameter indicates the value in seconds to which the user parameter **WaitTimeout** is initialized at the beginning of each job. A value of 0 indicates that the time-out is infinite. Trying to set the system parameter **WaitTimeout** to a negative value is ignored and the previous setting of **WaitTimeout** is used.

Legal values: 0 or any positive integer.

Errors: **typecheck**

3.5 Device Parameters

Device parameters are set using the operator **setdevparams** and are read using the operator **currentdevparams**. Device parameters are similar to system parameters in that they require a password (if the **SystemParamsPassword** is set), are global to the PostScript language environment, and persist across jobs. As with system parameters, some of these parameters may be stored persistently in non-volatile memory.

Device parameters are subdivided into sets that correspond to a particular device (for example, %Serial%, %disk2%). More generally, “device” in this context really means “named parameter set.” Each named parameter set known to the **currentdevparams**/**setdevparams** operators corresponds to an instance of the **IODevice** resource category and can represent a set of parameters describing the configuration of a physical or logical communications channel, storage device, hardware device, or software entity

such as a language emulator (see section C.4 of the *PostScript Language Reference Manual, Second Edition* for more details). Even if two products have the same named device, the parameters in the set might differ, for example, because the hardware support for that device differs on each product.

Note *Not all of the device parameters listed in tables 3.4 to 3.35 will be present in every printer product. Refer to the product addendum for a complete list of parameters supported by any given product.*

Device Parameter Dependencies

One property that distinguishes device parameters from both system and user parameters is that device parameters can be interdependent. The legality of a value for a given parameter may depend on the value of another parameter.

For example, in the serial communications device set there is an **Interpreter** and a **Protocol** parameter. The **Interpreter** parameter determines which page description language is to be used for an incoming job on that channel. The **Protocol** parameter determines the communications protocol used to send and receive data. **Protocol** can be set to /Binary, /Normal, /Raw, or /TBCP. The serial channel cannot be configured to have **Protocol** set to **/Raw** and **Interpreter** set to /PostScript. This would be an illegal combination of device parameters. This condition is termed a *configuration error.* A PostScript language error (**configurationerror**) occurs if **setdevparams** attempts to establish such an illegal configuration.

Most configuration dependencies are between parameters in the same device parameter set; there is a dependency among all communications devices, however, that requires at least one of the communications channels to be **On** and **Enabled**. There might also be cases where certain device parameter sets have interdependencies. For example, if both LocalTalk® and a serial channel share the same hardware port on a printer, there is a requirement that both never be **On** at the same time. If one channel is already **On** and the other is turned **On**, the first is turned off and disabled.

3.5.1 Device Parameter Set Types

Every device parameter set has a key-value pair which indicates its type. The key is **Type** and the value can be /Communications, /Emulator, /FileSystem, or /Parameters.

Table 3.3 *Parameters common to all device parameter sets*

Key	*Type*	*Semantics*
Type	name	*(Read-only)* This parameter designates the general category of parameters in a device parameter set. Every device parameter set shall contain a **Type** entry. **Legal values:** /Communications, /Emulator, /FileSystem, /Parameters **Errors:** None.

3.5.2 Device Parameters Associated with Communications

A raster output device can have various physical communications channels and can speak many different protocols over these channels. Host computers can communicate with these products by way of diverse network topologies and/or by way of direct point-to-point connections.

Communications Possibilities

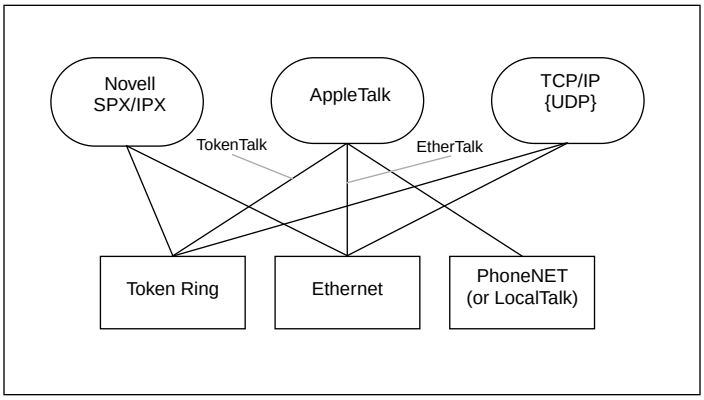
There are many ways to communicate with raster output devices. The choices of physical hardware for point-to-point communications include serial, unidirectional and/or bidirectional parallel, and SCSI bus. In the realm of network communications hardware, some choices are Ethernet, Token Ring, LocalTalk and PhoneNet®. There are network adaptors (e.g., line multiplexers, parallel-to-Ethernet adaptors and SCSI-to-Ethernet adaptors, among others) that allow the raster output device to connect to networks to which it does not otherwise have a direct physical connection. However, the network parameters described later in this chapter are relevant only to raster output devices that are true “peers” of the network with a direct connection to it. These products may support any of the popular protocol stacks, such as, TCP/IP, AppleTalk®, and the Novell® NetWare® (IPX®/SPX®). Although the design of most of these protocols predate the Open Systems Interconnection (OSI) Reference Model of the International Standards Organization (ISO), we shall use the OSI model in our description of the parameters associated with the functional layers of these protocol stacks.

There are various device parameter sets defined to aid system administrators in setting up and maintaining network printers. These parameter sets correspond to layers within the OSI model. Only the application layer (of OSI) possesses a parameter set of type /Communications. The PostScript interpreter (or emulator) receives its jobs from this layer. The device parameter sets, which correspond to the transport, network, data link and physical layers, are of type /Parameters. The parameter set associated with an implementation of the network layer must reference a parameter set associated with an instance of the data link/physical layer (in other words, a

distinct interface to the network). For example, if there is only one network layer implementation, but it is connected to *n* network interfaces, then for each network layer parameter set there is a unique data link/physical parameter set named within it. A parameter set for the network layer can be viewed as a distinct binding of the network address and the network interface.

There are various standard communications protocols that raster output devices must speak in order to reside directly on networks. The choices of protocols and the physical hardware medium on which they reside are limited in certain ways. Figure 3.1 illustrates the present relationship between these protocols and physical communications hardware. The arrows in this diagram indicate that the network protocols can be used with the physical medium pointed to in order to deliver and receive messages.

Figure 3.1 *Relationship between network communications protocols and physical communications medium*



The Novel Netware® communications protocols were originally derived from the Xerox Network System (XNS) protocols. For example, the Novell Internetwork Packet Exchange (IPX) is virtually identical to the Xerox network layer protocol called IDP (Internetwork Datagram Protocol). The Novell transport protocol is called Sequenced Packet Exchange (SPX)[1].

AppleTalk is the name Apple Computer, Inc. chose for their networking software that is built into every Macintosh computer. The data link protocol LocalTalk Link Access Protocol (LLAP) is used while communicating over LocalTalk or PhoneNET networks. (ELAP) EtherTalk Link Access Protocol and (TLAP) TokenTalk Link Access Protocol are the data link protocols for communication over Ethernet® and TokenRing, respectively. AppleTalk over Ethernet is called EtherTalk, and AppleTalk over Token Ring is called TokenTalk®.

1. Malamud, Carl: *Analyzing Novell® Networks*, Van Nostrand Reinhold, New York, page 81, © 1992.

TCP/IP is a network architecture sponsored by the Defense Advanced Research Project Agency (DARPA) and has been adopted by a large number of vendors. TCP stands for Transmission Control Protocol and is a transport layer protocol. IP stands for Internet Protocol and is the network layer protocol used by TCP. UDP (User Datagram Protocol) is a connectionless protocol. It is also dependent on IP for routing to the destination but does not ensure that the destination receives the packets.

Refer to *Computer Networks* by Andrew S. Tanenbaum[2] for an excellent description of Token Ring and Ethernet.

See *Inside AppleTalk*® for a description of AppleTalk.[3]

Before describing various communication parameter sets, let us first think more about how parameter sets of type /Communications are manipulated.

Communications Parameter Sets

There can be different ways to set up a product's communications parameters, including the use of front panels, hardware switches, and PostScript operators and procedures. The scheme described in this section provides a generic model for setting communications parameters. This model works across a variety of products and enables PostScript language spoolers and utilities to use the same model when reading and writing communication device parameters.

A raster output device typically has several hardware ports for communications. For example, a printer might have a parallel port and two serial ports named channel A and channel B. The parallel port is associated with the parameter device set named %Parallel%. Serial channel A, which is wired to a 25-pin RS-232A connector, is associated with the parameter device set named %Serial%. Serial channel B, which is wired to either an 8- or 9-pin connector, is associated with the parameter device set named %SerialB% or with the parameter set named %LocalTalk%. In this example, two device sets are associated with the same hardware port.

For any given communications device set, there are three sets of parameters. If the name of the device is %CommName%, the names of the three parameter sets are %CommName_NV%, %CommName%, and %CommName_Pending%. For example, in a printer with an SCC chip and a parallel port, the following parameter sets probably would be available:

---

2. Tanenbaum, Andrew S., *Computer Networks*, Prentice-Hall, Inc., New Jersey, © 1981.
3. Sidhu, Andrews, and Oppenheimer, *Inside AppleTalk*®, *Second Edition*, Addison-Wesley Publishing Company, Inc., Menlo Park, California and other locations, © 1990 by Apple Computer, Inc.

%Parallel_NV%	%Parallel%	%Parallel_Pending%
%Serial_NV%	%Serial%	%Serial_Pending%
%SerialB_NV%	%SerialB%	%SerialB_Pending%
%LocalTalk_NV%	%LocalTalk%	%LocalTalk_Pending%

The three parameter sets for a communications channel have the following general characteristics:

- %CommName_NV% values usually are stored in non-volatile memory.
- %CommName% values usually are stored in RAM and do not persist when the printer is powered off.
- %CommName_Pending% is a *read-only* parameter set whose values are used to configure the communications hardware and software at the beginning of the next file. This parameter set reflects either the current values of some writable parameter set, such as %CommName%, or some predetermined values selected via a switch or front panel. How the system computes the values in %CommName_Pending% is described below.
- When there are multiple instances of a certain communications parameter set, the naming convention is %CommName%, %CommNameB%, %CommNameC% and so on.

The name %CommName_NV% is only a hint of actual behavior. In products with limited non-volatile memory, only some of the %CommName_NV% set parameters may actually be saved to non-volatile memory, while products with sufficient non-volatile memory typically save all writeable %CommName_NV% parameters. PostScript language utility programs need not take these differences into account. If their specific intent is to affect persistent values, they should use %CommName_NV%. The implementation will do the best it can given the amount of non-volatile memory available in the product.

There is a hierarchical relationship between these parameter sets as described below. On some products these three sets may not be distinct from each other. The reason for the presence of the three sets on all products is to provide for a consistent model that is product-independent.

Basic Hierarchy of Parameter Sets

This description begins with a simple subset of the model and progresses to more complex situations.

Figure 3.2 *Relationship between the communication parameter sets*

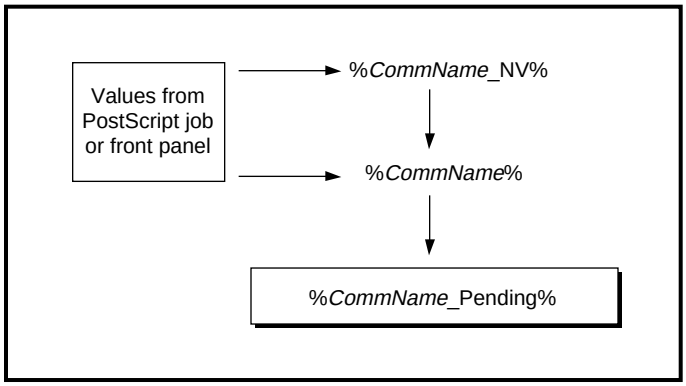


Figure 3.2 shows the basic relationship of the three parameter sets. In this figure, values written to %CommName% are written through to %CommName_Pending%; and values written to %CommName_NV% are written through to %CommName% and thence to %CommName_Pending%. Beyond this, several variables exist:

- The product may have a front panel. The values set by the user at the front panel are written to %CommName% or to %CommName_NV% (if the values are to persist across restarts and power cycles). Some products store to only one of these sets.

- The product may have switches through which it can be directed to use either %CommName_NV% parameter sets or built-in (hard-wired) values. (This situation is discussed at length later in this section.)

  Most products do not have both a front panel and switches.

- PostScript language programs (usually spoolers or utilities) may write parameter values to %CommName% or %CommName_NV% (usually the former) at any time. This is true whether the output device has a front panel or has switches.

In Figure 3.2, the %CommName% parameter set, which is in RAM and does not persist when the printer is powered off, is used in many cases (but not all) to update the %CommName_Pending% set. Thus, on many products (those with a front panel but no switches), the %CommName% and %CommName_Pending% sets always have the same values and appear redundant.

The %CommName_NV% set usually stores the parameters in non-volatile storage. In the simple case in Figure 3.2, writing to %CommName_NV% writes through to %CommName%, which in turn writes through to %CommName_Pending%.

In general, a spooler or utility almost always should write to %CommName%. It should write to %CommName_NV% only if parameters are to persist when the printer is turned off.

A front panel usually writes to %CommName_NV% to change the power-on parameters, although the front panel also can write to %CommName%.

Multiple Non-Volatile Sets

Complicating this picture, it is possible to have more than one non-volatile parameter set. Such sets are correctly named as follows: %CommName_NV%, %CommName_NV2%, %CommName_NV3% and so on. As is the case with a single non-volatile set, these parameter sets obtain their values by being written to by a PostScript language spooler or utility.

Figure 3.3 *Communications parameters sets using NV values*

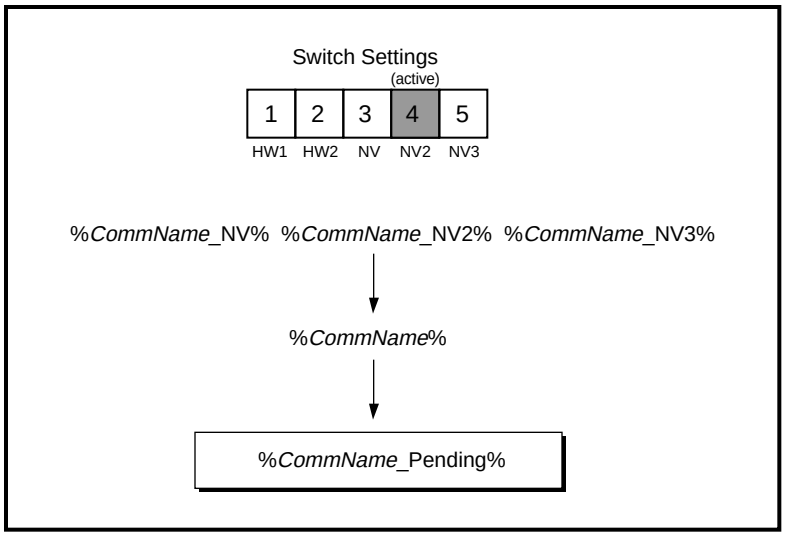
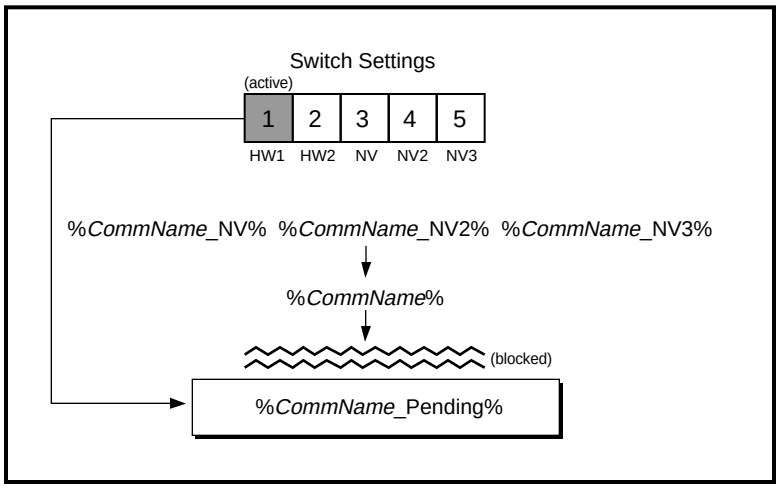


Figure 3.1 shows a situation in which there are three non-volatile sets. Only one of these sets can be active at any given time. The switch setting indicates which one is active. In this figure, the active set is %CommName_NV2%, which is indicated by the switch setting. When the switch is set to this position, or when the product is restarted or powered up with the switch in this position, the values in %CommName_NV2% are written through to %CommName% and to %CommName_Pending%. While the setting %CommName_NV2% is active, a PostScript language job can write to any of the non-volatile parameter sets, but only if it wrote to %CommName_NV2% would the values migrate to %CommName% and %CommName_Pending%. Changing the switch to the position corresponding to %CommName_NV3% would cause %CommName_NV3% values to become the active ones in %CommName% and %CommName_Pending% instead.

Predetermined Parameter Values

In addition to the switch settings that indicate which non-volatile parameter set should be used, there can also be other switch settings that short-cut this hierarchy of parameter sets and cause a predetermined set of communications parameters to be written directly to %CommName_Pending%. This situation is shown in Figure 3.4.

Figure 3.4 *Communications parameter sets using "hard wired" values*



In the figure, switch positions 1 and 2 designate two such "hard-wired" parameter sets. When the switch is set to position 1, for example, PostScript language programs may still write to one of the %CommName_NV% sets or to %CommName%, but there is no effect on %CommName_Pending% unless the switch is reset to one of positions 3 through 5.

This example explains the existence of the %CommName_Pending% set as separate from the %CommName% set: it allows absolute determination of the communication parameters that will be used, no matter what other activity occurs.

Note *Reading the %CommName_NV% set or the %CommName% set gives you no information about the parameters being used for the current job or the next job, but simply returns the values last written to these sets. Reading %CommName_Pending% returns the values to be used for the next job. Determining the parameters of the current job is of little interest. Either the job is a page description, in which case it should not be accessing device parameters at all, or the job is a utility that is interested in either determining or affecting the settings for future jobs. If the device parameters are used as described above, utilities can be written without concern for exactly which parameters are stored in non-volatile memory and without concern for whether a utility job, front panel, or switch is used to establish communication parameters.*

As in the case described in the previous section, a spooler or utility almost always should write to %CommName%. It should write to %CommName_NV% only if parameters are to persist across restarts and power cycles.

Changes to parameters of type /Communications take effect after the current file (containing one or more PostScript language jobs) is fully processed by the interpreter and prior to reading from the next file.

A file which specified communications changes will complete before a transition to new settings (of the %CommName_Pending% set) takes place. The user must ensure that no additional data will be transferred from host to printer on this communication channel until after the transition to new settings is complete.

Note *There is a distinction between "end of job" and "end of file." There can be multiple jobs in a file. See section 3.7.7, "Job Execution Environment," in the PostScript Language Reference Manual, Second Edition.*

Parameters Present in Parameter Sets of Type /Communications

The parameters listed in Table 3.4 on page 66 are those found in all device parameter sets of type /Communications.

Table 3.4 *Parameters present in parameter sets of type /Communications*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	This parameter selects how the output channel is managed after each job finishes executing. The printer does not wait for the pages of the job to finish printing, but instead immediately starts executing the next job. The **DelayedOutputClose** boolean parameter is set independently for each communications channel. When **DelayedOutputClose** is true: • An end-of-file (EOF) is not sent until all pages of a job have been printed. On network channels, the connection remains open until the job finishes printing. • If a job produces output, and if there are preceding jobs that have not finished printing that are using the same output channel, the output will not be sent until those jobs have completed printing and the EOFs for them have been sent. • Spontaneous messages, such as printer error messages, are sent to the channel if it is either the output channel for the job executing or the output channel for jobs that have finished executing but have not finished printing.

When **DelayedOutputClose** is *false*:

- An EOF is sent as soon as a job finishes executing in the interpreter. On network channels, the connection may be closed when the job finishes executing, even though pages produced by the job might not have finished printing.
- Output generated by a job can be transmitted without delay, even if there are previous jobs using the same output channel that have not finished printing (the EOF for those jobs will have already been sent).
- Spontaneous messages, such as printer error messages, are sent to the channel only if it is the output channel for the job executing, even if it is the output channel for previous jobs that have not finished printing.

The **DelayedOutputClose** setting for a job source is controlled by the parameter sets for the output channel of that job source. So, if the serial B channel is used for the output of jobs received on the parallel port, then the **DelayedOutputClose** value in the %SerialB% parameter set applies to jobs received on both the serial B and parallel ports.

The **DelayedOutputClose** parameter does not appear in a communications parameter set if the channel has no output or if all messages generated asynchronously from the interpreter are directed to a logically separate channel.

Note *In versions prior to 2014, upon completion of each job, the interpreter waits for all pages of the job to be printed before sending an EOF or closing a connection and before starting to execute another job. Thus, any output associated with the job, including printer error messages, is always sent before the next job begins and before sending the EOF or closing a connection.*

Legal values: *true*, *false*

Errors: None.

Enabled boolean This parameter designates whether data arriving on the communications channel represented by the parameter set should be considered as a job to be scheduled for execution by the PostScript interpreter or an emulator. If **Enabled** is *true*, arriving data is scheduled as an executable job. If **Enabled** is *false*, the data will not be scheduled as an executable job, but the channel can be used directly by a job for reading and writing data. A **configurationerror** is generated if setting **Enabled** would produce either of the following situations:

- When trying to set **On** to *false* and **Enabled** to *true* within the same parameter set, a **configurationerror** will result.

- When trying to turn off **Enabled** in one communications device parameter set results in all channels having **Enabled** set to *false*, a **configurationerror** will result.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Filtering name This parameter indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language.

Legal values: /InterpreterBased, /None

Errors: **configurationerror**, **rangecheck**, **typecheck**

/InterpreterBased: In this mode, the input stream is filtered as necessary to conform to the language. For example, the data stream may have been sent to the printer encoded as a TBCP PostScript language job and must be decoded to a normal PostScript language job before it is passed to the interpreter (see **Protocol** in Table 3.5 on page 71 for a description of TBCP).

/None: Pass the data unchanged to the interpreter.

Warning *In a complete AppleTalk/Macintosh environment*, **Filtering** *should be set to /None or you will encounter communications problems.*

HasNames boolean *(Read-only*) This parameter indicates whether the communications channel represented by the parameter set supports named files. **HasNames** is always *false* in device parameter sets of type /Communications. This parameter is defined only in device parameter sets of type /FileSystem or /Communications.

Legal value: *false*

Errors: None.

Interpreter name This parameter designates which interpreter or emulator is to be used to interpret the next incoming job arriving on this communications channel. This parameter is used only if **Enabled** is *true* and **PrinterControl** is /PSPrinter. For certain communication channels there is a relationship between the **Interpreter** and the **Protocol** parameters that can result in a **configurationerror**. See **Protocol** in Table 3.5 on page 71 for further details.

Either **Interpreter** or **Protocol** or both can be set without a password if no other parameters are specified in the execution of **setdevparams**.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

The **Interpreter** value /AutoSelect is described below. For information on the other legal values, see section 3.5.9, “Emulator Parameters.”

/AutoSelect: The AutoSelect facility provides automatic and seamless switching between the available interpreters and emulators based on the input data stream. The **Interpreter** parameter should be set to /AutoSelect on channels that connect to hosts which alternately send PostScript language jobs, raw PCL® (LaserJet IIP or LaserJet III) and “printscreen” jobs (in the IBM® PC compatible environment). It can be used on any communications channel. When using AutoSelect for a given communications channel, it is important that the underlying communications protocol is one that preserves all incoming data. In particular, for a serial or parallel channel, this implies that **Protocol** is set to /Raw, /Binary, or /TBCP.

For serial and parallel communication channels, the following is true:

- AutoSelect detects interpreter boundaries and job boundaries if the value of **Protocol** is set to /TBCP or /Binary.
- AutoSelect detects interpreter boundaries, job boundaries, and protocol boundaries and automatically selects the protocol if the value of **Protocol** is set to Raw. This is the recommended setting for **Protocol** when using AutoSelect. When AutoSelect detects that a PostScript language job is being received and the **Protocol** is Raw, only the Normal and TBCP protocols can be recognized (e.g., Binary is not supported).
- When **Interpreter** is set to /AutoSelect, the value of **Protocol** must be /Binary, /Raw, or /TBCP; otherwise, a **configurationerror** is generated.

For other communication channels that are binary in nature, the following is true:

- The /PCL value is used only in printers that emulate a LaserJet 4 or later LaserJet printers. For emulations of earlier LaserJets, the values /LaserJet II and /LaserJetIII are used.
- AutoSelect detects interpreter boundaries and job boundaries.

On boolean This parameter designates whether the communication channel is turned on and able to receive and send data. If the parameter is *true*, data transmitted to the channel by a host is buffered and flow control protocols are applied. Data sent to the channel when this parameter is *false* is lost. A **configurationerror** is generated if setting the **On** parameter would produce a situation in which **On** is *false* and **Enabled** is *true* in the same parameter set.

If two communication devices share the same physical port, and setting the **On** parameter produces a situation in which both channels had **On** set to *true*, the one that was originally **On** is turned off and disabled, and the new one is turned **On**.

If **On** is *true* and **Enabled** is *false*, the channel is not considered as a source of jobs to be scheduled, but the channel can be used by a PostScript language job to send and receive data by means of the file operators.

During power up, if it is determined that all installed communication channels are currently off, it is up to the product to perform its own unique recovery strategy. For example, the product could search for an installed communications channel and force it on even if this was not the state preserved in non-volatile memory. Another alternative would be to inform the user via the operator control panel that the product cannot be initialized until the problem is rectified.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name This parameter is used to select or indicate how a host queries and controls the printer for the communication channel associated with this parameter set.

Legal values: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

If **/PrinterControl** is set to /PSPrinter, the following statements are true:

- The **Interpreter** parameter selects the page description language.
- Printer error messages are sent in usual Adobe fashion (on channels processing jobs, and in `%%[ . . . ]%%` format).
- PJL (Printer Job Language) commands are not recognized unless the **Interpreter** parameter is /AutoSelect or /PCL. If the Interpreter parameter is /AutoSelect, `ENTER LANGUAGE` and `UEL` are handled, and other PJL commands are identified as PJL commands and discarded. If **Interpreter** is /PCL, `UEL` is handled.

If **/PrinterControl** is set to /PJL, the following statements are true:

- PJL (Printer Job Language) controls language selection.
- Printer error messages are in PJL format and are enabled or disabled by PJL commands. Whether a job is being processed on a channel does not affect whether messages are sent.
- All LaserJet 4 PJL commands are handled as they would be on a LaserJet 4 or 4si printer.
- The “PJL current environment” is used on each invocation of a page description language to setup the initial state.

The PJL language is described in the HP Manual called *Printer Job Language Reference Manual*, (Manual Part No. 5961-0998). The edition of this manual published in May, 1993 includes the PJL support on LaserJet 4si (4si MX) printers. Earlier editions describe the PJL on LaserJet 4 (and 4M) printers, but not the LaserJet 4si printers.

Full LaserJet 4 emulation is selected for a channel by setting **PrinterControl** to /PJL. Users can also select PCL5e interpretation, without full LaserJet 4 emulation, by setting **PrinterControl** to /PSPrinter and setting Interpreter to /PCL. In this mode, all input is assumed to be PCL5e.

Serial Communications Parameters

Table 3.5 on page 71 lists those parameters typically found in the device parameter sets named %Serial%, %SerialB%, %SerialC% and so on.

Table 3.5 *Parameters present in %Serial% communications parameter sets*

Key	*Type*	*Semantics*
Baud	integer	This parameter designates the baud rate on the underlying serial hardware. Normally this parameter can be set to any non-negative number; it will not be rounded. The underlying serial hardware will, however, round the baud rate to the nearest achievable value. Hardware rounding will not be reflected in the value of the parameter when it is read. On some products this parameter might be restricted to a small number of legal values. **Legal values:** Product-dependent. **Errors:** **rangecheck**, **typecheck**

CheckParity boolean This parameter designates whether parity checking is done by the device on incoming data. This parameter is ignored if the value of **Parity** is /None. If **CheckParity** is *true* and a parity error occurs, a PostScript language **ioerror** results. If **CheckParity** is *false*, no parity checking occurs.

Legal values: *true*, *false*

Errors: **typecheck**

DelayedOutputClose
boolean For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66.

Legal values: *true, false*

Errors: None.

DataBits integer This parameter designates the number of data bits per byte communicated over the channel. If the value of this parameter is 7, the high bit of a received byte of data is set to 0. The total number of bits for each byte transmitted or received is the sum of the number of start bits (always 1), data bits, parity bits and stop bits.

Legal values: 7, 8

Errors: **rangecheck**, **typecheck**

Enabled boolean For the general definition of **Enabled**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

FlowControl name This parameter designates the serial flow control method used between the host and the device**.**

Note *Not all serial channels support all flow control modes.*

Following are descriptions of the legal values of **FlowControl**:

/Dtr: DTR (Data Terminal Ready) and DSR (Data Set Ready) hardware signals are used by the printing device and the host, respectively, to indicate to the other when data may be transmitted. A high value for the signal indicates that data may be transmitted, a low value indicates that data should not be transmitted.

/DtrLow: This parameter is the same as /Dtr except the active sense of the signals is reversed. A low signal indicates that data may be transmitted, a high signal indicates that data should not be transmitted.

/EtxAck: Two characters, ETX (end-of-text) and ACK (acknowledgment), are reserved for flow control usage. The protocol is symmetric for printing device and host. Each sender knows an agreed upon maximum number of characters that the other side can receive. A sender may send up to this number of characters followed by an ETX. The sender may send more data only when it has received an ACK from the receiver on the other side.

/RobustXonXoff: This protocol operates similarly to the /XonXoff protocol, except that periodically (typically every second) the interpreter will send the host an Xon if it is able to receive data.

/XonXoff: This protocol is used by PCs. Two characters, XON and XOFF, are reserved for flow control usage. For all **Protocol** settings except /Raw, the protocol is symmetric for printing device and host. If one side wishes the other to stop sending data, it sends an XOFF. When it is ready to receive data again it sends an XON. When the **On** parameter is set to *false*, the interpreter sends an /XOFF to the host just before turning off the channel. If **Protocol** is set to /Raw, XON and XOFF sent from the host to the printer are treated as data and not reserved as flow control characters. XON and XOFF sent from the printer to the host are to be treated as flow control characters.

/XonXoff2: This protocol is used by UNIX systems. It operates similarly to the /XonXoff protocol except that when the **On** parameter is set to false, the interpreter sends an /XON to the host. This allows the host to unload any data it wishes to send. The interpreter simply drops the data on the floor.

Legal values: /Dtr, /DtrLow, /EtxAck, /RobustXonXoff, /XonXoff, /XonXoff2

Errors: **rangecheck**, **typecheck**

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Parity name This parameter designates the parity to be used between the host and the device. If **Parity** is /Space or /Mark, the parity bit should always be 0 or 1, respectively. If **Parity** is /None, neither the host nor the device should send a parity bit. If **Parity** is /Even, even parity is used. If **Parity** is /Odd, odd parity is used. The total number of bits for each byte transmitted or received is the sum of the number of start bits (always 1), data bits, parity bits and stop bits. Most serial devices do not support 8-bit data with either space or mark parity, although setting the parameters in this manner does not generate a **configurationerror**. The results of this configuration, however, are unpredictable.

Legal values: /Even, /Mark, /None, /Odd, /Space

Errors: **rangecheck**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Protocol name This parameter indicates the communications protocol that is used.

Legal values: /Binary, /Normal, /Raw, /TBCP

Errors: **configurationerror**, **rangecheck**, **typecheck**

/Binary: In /Binary mode, an encoding scheme allows the full range of 8-bit values to be transmitted as data while also providing for certain communications functions, such as end-of-file, software flow control, abort job, status query and so on. This protocol is suitable for use with any language (for example, the PostScript language or a printer emulation). However, it is obsolete and has been superseded by /TBCP.

/Normal: In /Normal mode, certain control characters are reserved as communications functions, such as end-of-file, software flow control, abort job, status query and so on. These codes cannot be carried as data. This protocol is suitable for use only when sending ASCII-encoded PostScript language jobs; it is unsuitable for PostScript language jobs containing binary data or any printer emulation jobs.

/Raw: In /Raw mode, all characters are treated as data; there are no reserved characters, and one of the communications functions is available. Normally, this protocol is suitable for use only with printer emulation, not with the PostScript interpreter. However, in products that support an **Interpreter** value of /AutoSelect, protocol processing is handled by the AutoSelect facility; therefore, **Protocol** should be /Raw when **Interpreter** is set to /AutoSelect.

/TBCP: In /TBCP (Tagged Binary Communication Protocol) mode, an encoding scheme allows the full range of 8-bit values to be transmitted as data, while also providing for certain communication functions, such as end-of-file, software flow control, abort job, status query and so on. It also provides explicit begin-protocol and end-protocol sequences that permit the receiver to switch automatically between /Normal and /TBCP mode processing. This protocol is suitable for use with any language (for example, the PostScript interpreter or a printer emulation).

For more information on these protocols, refer to technical note #5009 titled *Adobe Serial and Parallel Communications Protocols Specification*, available from the Adobe Developers Association.

A **configurationerror** is generated if setting either the **Protocol** or the **PrinterControl** parameters would result in the following combination:

- **Protocol** with a value of /Normal and **PrinterControl** with a value of /PJL.

A **configurationerror** is also generated if setting the **Protocol** or **Interpreter** parameter would produce one of the following situations when the **Enabled** parameter is *true* and the **PrinterControl** parameter is /PSPrinter:

- **Protocol** with a value of /Raw and **Interpreter** with a value of /PostScript.
- **Protocol** with a value of /Normal and **Interpreter** with a value other than /PostScript.
- **Protocol** with a value of /Normal and **Interpreter** with a value of /AutoSelect.

That is, PostScript language jobs cannot be executed over a channel using the /Raw protocol, and emulators cannot be executed over a channel using the /Normal protocol. Likewise, when doing automatic selection of interpreters and emulators, the /Normal protocol cannot be used.

Either **Protocol** or **Interpreter** or both can be set without a password if no other parameters are specified in the execution of **setdevparams**.

StopBits integer This parameter designates the number of stop bits that are transmitted by the serial hardware. The hardware will always be able to receive data transmitted with one or two stop bits. The total number of bits for each byte transmitted or received is the sum of the number of start bits (always 1), data bits, parity bits, and stop bits.

Legal values: 1, 2

Errors: **rangecheck**, **typecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

Parallel Communication Parameters

Table 3.6 on page 76 lists those parameters typically found in the device parameter sets named %Parallel%, %ParallelB%, %ParallelC% and so on.

Table 3.6 *Parameters present in %Parallel% communications parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of DelayedOutputClose, see Table 3.4 on page 66.
	Note	**DelayedOutputClose** *should only be present in this set if bidirectional communications is possible.*
	Legal values:	*true*, *false*
	Errors:	None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66.
	Legal values:	*true*, *false*
	Errors:	**configurationerror**, **typecheck**
Handshake	integer	This parameter indicates the hardware/software signal interface that is to be used for communications across the parallel interface. If this key is not present, the default is unidirectional parallel.

Note *"The IEEE-1284 Draft Specification 2.00" refers to the document "Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers, IEEE P1284 D2.00, September 10, 1993."*

Legal values: 0, 1, 2, 3, 4, 5, 6, 7, 8

Errors: **configurationerror**, **rangecheck**

0	Unidirectional communications commonly used by PCs and PC-compatibles.
1	Bidirectional communications as specified by version 0.6 of the Hewlett-Packard Boise specification.
2	IEEE-1284 Draft Specification 1.00.
3	Unidirectional, Ack in Busy.
4	Unidirectional, Ack after Busy (Japan implementation).
5	Unidirectional, Ack while Busy.
6	IEEE-1284 Draft Specification 2.00, Ack in Busy.
7	IEEE-1284 Draft Specification 2.00, Ack after Busy.
8	IEEE-1284 Draft Specification 2.00, Ack while Busy.

Values 1 and 2 are obsolete. Value 1 is superseded by value 2 which is superseded by values 6, 7 and 8.

Setting the **OutputDevice** key to %Parallel% will generate a **configurationerror** when the **Handshake** key is set to one of the unidirectional values (0, 3, 4, 5). Conversely, if the **OutputDevice** key is set to %Parallel%, then setting the **Handshake** key to one of the unidirectional values will generate a **configurationerror**.

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal values: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

nAckPulseWidth integer There is a signal which originates from the peripheral to the host called *nAck*. This signal is in the form of a pulse and its meaning depends upon which mode of operation is being used or which state the peripheral device driver is currently in. The pulse width is controlled by the peripheral support circuitry or device driver. This parameter allows one to examine or change the *nAck* pulse width. The value is the number of nanoseconds for the duration of the pulse (rounded to nearest value that can be achieved).

Legal values: Normally an integer in the range of 500 to 10000.

Errors: **rangecheck**, **typecheck**

nStrobeExpectedPulseWidth integer There is a signal which originates from the host to the peripheral called *nStrobe*. This signal is in the form of a pulse and its meaning depends upon which mode of operation is being used or in which state the peripheral device driver is currently. The pulse width may vary from host to host. This parameter allows one to examine or change the expected duration of the *nStrobe* pulse width. The value is the number of nanoseconds for the duration of the pulse (rounded to nearest value that can be achieved).

Legal values: Normally an integer in the range of 750 to 500000.

Errors: **rangecheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

OutputDevice string This parameter specifies which communications device is to be used for %stdout and %stderr. If the value of **OutputDevice** is the empty string, %stdout% and %stderr% information is routed to the default back channel specified for the device.

Setting the **OutputDevice** key to %Parallel% will generate a **configurationerror** when the **Handshake** key is set to one of the unidirectional values (0,3,4,5). Conversely, if the **OutputDevice** key is set to %Parallel% then setting the **Handshake** key to one of the unidirectional values will generate a **configurationerror**.

Legal values: %Serial%, %SerialB%, %SerialC% and so forth; %Parallel%, %ParallelB%, %ParallelC% and so forth; or the empty string.

Errors: **rangecheck**, **configurationerror**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Protocol name For the general definition of **Protocol**, see Table 3.5 on page 71.

Legal values: /Binary, /Normal, /Raw, /TBCP

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

SCSI Communications Parameters

Table 3.7 on page 79 lists those parameters typically found in the device parameter sets named %ScsiComm%, %ScsiCommB%, %ScsiCommC% and so on.

Table 3.7 *Parameters present in %ScsiComm% communications parameter sets*

Key	*Type*	*Semantics*
Bus	string	(*Read-only*) This parameter designates which SCSI bus device parameter set is associated with this %ScsiComm% channel. **Legal values:** %Scsi%, %ScsiB%, %ScsiC%, and so forth **Errors:** None.

DelayedOutputClose

boolean — For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: None.

Enabled boolean — For the general definition of **Enabled**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Filtering name — For the general definition of **Filtering**, see Table 3.4 on page 66.

Legal values: /Interpreter, /None

Errors: **configurationerror**, **rangecheck**, **typecheck**

HasNames boolean — (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name — For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

On boolean — For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name — For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

LocalTalk Communications Parameters

Table 3.8 on page 81 lists those parameters typically found in the device parameter sets named %LocalTalk%, %LocalTalkB%, %LocalTalkC% and so on.

Table 3.8 *Parameters present in %LocalTalk% communications parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerro**r, **typecheck**
Filtering	name	For the general definition of **Filtering**, see Table 3.4 on page 66. **Legal values**: /Interpreter, /None **Errors**: **configurationerror**, **rangecheck**, **typecheck**
HasNames	boolean	(*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66. **Legal value:** *false* **Errors:** None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

LocalTalkType string This parameter represents the *type* piece of the AppleTalk *entity name*. The entity consists of three pieces: *zone*, *type*, and *object*, each of which is a string of 32 or fewer non-null characters. The *object* piece is set to the value of the **PrinterName** system parameter and the *zone* is set to the wildcard character (asterisk).

If the printer also supports EtherTalk and/or TokenTalk communications, setting the **LocalTalkType** string will set the **EtherTalkType** and/or **TokenTalkType** parameter to the same value. The **appletalktype** compatibility operator will reflect a change to the **LocalTalkType** parameter. Therefore, getting the **LocalTalkType** parameter will always yield the same value as getting the **EtherTalkType** and/or the **TokenTalkType** parameter and will match what is returned by the **appletalktype** compatibility operator.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **limitcheck**, **typecheck**

NodeID integer (*Read-only*) This parameter represents the local network address of the device. Legal addresses are 0 or values between 128 to 254 inclusive. If the value of **NodeID** is 0, this indicates that the address has not been established. The value is used as an address hint when first establishing addresses as part of the LocalTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the LocalTalk device.

Legal values: 0 or any integer between 128 and 254 inclusive.

Errors: None.

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

EtherTalk Communications Parameters

Table 3.9 on page 83 lists those parameters typically found in the device parameter sets named %EtherTalk%, %EtherTalkB%, %EtherTalkC% and so on.

Table 3.9 *Parameters present in %EtherTalk% communications parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
EthernetAddress	string	(*Read-only*) This parameter is a unique string that represents the Ethernet address of the printer. The string is of the form (xx:xx:xx:xx:xx:xx), where each x represents a digit in hexadecimal. **Legal values:** A string of 17 characters representing a legal Ethernet address. **Errors:** None.
EtherTalkType	string	This parameter represents the *type* piece of the EtherTalk *entity name*. The *entity name* consists of three pieces: *zone*, *type*, and *object*, each of which is a string of 32 or fewer non-null characters. The *object* piece is set to the value of the **PrinterName** system parameter. The *zone* is set to the printer zone name. If the printer also supports LocalTalk and/or TokenTalk communications, setting the **EtherTalkType** string will set the **LocalTalkType** and/or **TokenTalkType** parameter to the same value. The **appletalktype**

compatibility operator will reflect a change to the **EtherTalkType** parameter. Therefore, getting the **EtherTalkType** parameter will always yield the same value as getting the **LocalTalkType** and/or the **TokenTalkType** parameter and will match what is returned by the **appletalktype** compatibility operator.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **typecheck**

EtherTalkZone string This parameter represents the *zone* piece of the EtherTalk *entity name*.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **typecheck**

Filtering name This parameter indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language. For the general definition of **Filtering**, see Table 3.4 on page 66.

Warning *In a normal network environment, **Filtering** should be set to /None or you will encounter communications problems.*

Legal values: /InterpreterBased, /None

Errors: **configurationerror**, **rangecheck**, **typecheck**

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

NodeID integer (*Read-only*) This parameter represents the local network address of the device. Legal addresses are values between 1 to 254 inclusive. If the value of **NodeID** is 0, this indicates that the address has not been established. The value is used as an address hint when first establishing addresses as part of

the EtherTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the EtherTalk device.

Legal values: Any integer between 0 and 254 inclusive.

Errors: None.

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

TokenTalk Communications Parameters

Table 3.10 on page 85 lists those parameters typically found in the device parameter sets named %TokenTalk%, %TokenTalkB%, %TokenTalkC% and so on.

Table 3.10 *Parameters present in %TokenTalk% communications parameter sets*

Key	*Type*	*Semantics*
Address	string	(*Read-only*) This parameter is a unique string that represents the TokenRing address of the unit. The string is of the form (*xx*:*xx*:*xx*:*xx*:*xx*:*xx*), where each *x* represents a digit in hexadecimal. **Legal values:** A string of 17 characters representing a legal address. **Errors:** None.
Bridging	name	Bridging, on the token ring, can be done in several different ways. When this parameter is set to /Transparent, this implies a transparent bridging where the entire “universe” is one large single ring structure and all identities are unique. When set to /SourceRoute, routing is done via specifying an explicit

path including the ring identification, RIF. When set to the default value /Adaptive, the software will automatically recognize the routing style and respond in kind (either as a one-time determination or when processing each connection).

Legal values: /Transparent, /SourceRoute, /Adaptive

Errors: **configurationerror**, **typecheck**

DelayedOutputClose

boolean For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: None.

Enabled boolean For the general definition of **Enabled**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Filtering name This parameter indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language. For the general definition of **Filtering**, see Table 3.4 on page 66.

Warning *In a normal network environment, **Filtering** should be set to /None or you will encounter communications problems.*

Legal values: /InterpreterBased, /None

Errors: **configurationerror**, **rangecheck**, **typecheck**

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

NodeID integer (*Read-only*) This parameter represents the local network address of the device. Legal addresses are values between 1 to 254, inclusive. If the value of **NodeID** is 0, this indicates that the address has not been established. The value is used as an address hint when first establishing addresses as part of the TokenTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the TokenTalk device.

Legal values: An integer between 0 and 254 inclusive.

Errors: None.

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true, false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

TokenTalkType string This parameter represents the *type* piece of the TokenTalk *entity name*. The *entity name* consists of three pieces: *zone*, *type* and *object*, each of which is a string of 32 or fewer non-null characters. The *object* piece is set to the value of the **PrinterName** system parameter. The *zone* is set to the printer zone name.

If the printer also supports LocalTalk and/or EtherTalk communications, setting the **TokenTalkType** string will set the **LocalTalkType** and/or **EtherTalkType** parameter to the same value. The **appletalktype** compatibility operator will reflect a change to the **TokenTalkType** parameter. Therefore, getting the **TokenTalkType** parameter will always yield the same value as getting the **LocalTalkType** and/or the **EtherTalkType** parameter and will match what is returned by the **appletalktype** compatibility operator.

Legal values: Any string of 32 or fewer non-null characters.

Errors: **typecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

Zone string This parameter represents the *zone* piece of the TokenTalk *entity name*.

Legal value: Any string of 32 or fewer non-null characters.

Errors: **typecheck**

OSI Application Layer Communications Parameters

This section describes those device parameter sets that correspond to the application protocol layer. This is the layer to which the PostScript interpreter (or language emulator) attaches for the purpose of receiving jobs and sending data back to the host. Certain parameter sets have been defined by Adobe Systems, Inc. which allow for the use of the TCP/IP protocol over Ethernet. These are %LPR%, %AppSocket% and %Telnet%. There are also parameter sets associated with the Novell Netware application layer. They are %RemotePrinter% and %PrintServer%. Each of these sets identifies a unique job source device for the PostScript interpreter or language emulator.

Node Address

Before listing the various network communications parameter sets, we must define the term *node address*. A node address is a unique address for a node on some network. The node address is in the form appropriate for the protocol being used to communicate with the node. The following table lists the various forms that a node address can take.

Table 3.11 *Node address forms*

Protocol	*Node address forms*	*Description*
TCP/IP	*N.N.N.N*	Each *N* is a decimal number in the range 0 to 255.
Novell SPX/IPX	*XXXXXXXX:xxxxxxxxxxxx*	Each *X* and each *x* is a hexadecimal digit in the range 0 to *F* (upper or lower case) *XXXXXXXX* is the network part of the address and *xxxxxxxxxxxx* is the *Media Access Control* part known as the *Novell Node Number.*
AppleTalk DDP	*N.N.n*	Each *N* and each n are decimal numbers in the range of 0 to 25. *N.N* represent the network part of the address and n represents the node ID.

LPR

The UNIX command `lpr` has the effect of sending a printer job to a printer. On the printer side, the LPR device name is used as the job source for incoming `lpr` jobs. There is an %LPR% device parameter set, and it is described in Table 3.9 on page 83. TCP port 515 is used for LPR. Because LPR (or `lpr` daemon) is by definition unidirectional, any %stdout or %stderr information is transmitted by means of the Syslog facility described in "Syslog" on page 104. The LPR service depends upon the TCP/IP protocol.

Table 3.12 *Parameters present in the %LPR% communications parameter set*

Key	*Type*	*Semantics*
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
Filtering	name	This parameter indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language. For the general definition of **Filtering**, see Table 3.4 on page 66.
	Warning	*In a normal network environment,* ***Filtering*** *should be set to /None or you will encounter communications problems.*
		Legal values: /InterpreterBased, /None **Errors:** **configurationerror**, **rangecheck**, **typecheck**
HasNames	boolean	(*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66. **Legal value:** *false* **Errors:** None.
Interpreter	name	For the general definition of **Interpreter**, see Table 3.4 on page 66. **Legal values:** /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors:** **configurationerror**, **rangecheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

PrintHost string This parameter is a list of at most two IP mask/address pairs where the mask is applied to the given IP address to specify which hosts are allowed to make LPR connections. The slash is used as a delimiter between the two subfields. If two address pairs are specified, they are separated by a space delimiter. The mask, which has the same syntax as the IP address, is optional. If an address is specified with no corresponding mask, a mask of 255.255.255.255 is assumed.

A mask/address pair in which the mask is not specified, for example, would be:

138.46.24.37

Two IP mask/address pairs would have the format:

255.255.255.0/138.46.24.37 255.255.255.0/138.46.24.38

Legal values: An empty string or a string (of 63 or fewer non-null characters) which specifies up to 2 IP mask/addresses separated by the ASCII blank character. An IP address can be of the form *N.N.N.N* where each *N* is a decimal number in the range 0 to 255. IP addresses cannot be set to illegal values (e.g., trying to use an IP address equal to 0.0.0.0, 127.0.0.0, 255.255.255.255, *N.N.N*.255 or other illegal values will result in a **rangecheck** error).

Errors: **typecheck**, **limitcheck**, **rangecheck**

ReceiveWindowSize

integer Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this

parameter states in order to accommodate the host's expectations. The receive window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%.

Legal value: An integer in the range from 1024 to 65535.

Errors: **typecheck**, **rangecheck**

SendWindowSize integer Specifying the send window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations. The send window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%.

Legal value: An integer in the range from 1024 to 65535.

Errors: **typecheck**, **rangecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

AppSocket

AppSocket was created to support TranScript™ software from Adobe Systems, Inc. It provides a more robust interface than LPR because it utilizes bidirectional communications directly. The AppSocket protocol can be used by drivers other than TranScript and for transmitting data other than PostScript language jobs.

Table 3.13 *Parameters present in the %AppSocket% communications parameter set*

Key	*Type*	*Semantics*
ControlPortNumber	integer	This parameter denotes a port used by the unit for the purpose of handshaking between the host and the unit while setting up a session. A session with the printer prevents other hosts from being able to interrupt the printer to run other jobs. Communications is via TCP, not UDP. The suggested default value is 9101. **Legal values:** A positive integer representing a port number not reserved by any of the standard services. **Errors:** **typecheck**, **rangecheck**, **configurationerror**
	Warning	*An error is not raised when a port number previously reserved for some other purpose is specified.*
DataPortNumber	integer	This parameter denotes a bidirectional port for transmission of printer language jobs. The suggested default value is 9100. Users are free to use another port number to avoid a conflict with another unit on the network already using 9100. **Legal values:** A positive integer representing a p ort number not reserved by any of the standard services.
	Warning	*An error is not raised when a port number previously reserved for some other purpose is specified.* **Errors:** **typecheck**, **rangecheck**, **configurationerror**
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
Filtering	name	This parameter indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language. For the general definition of **Filtering**, see Table 3.4 on page 66.

Warning *In a normal network environment, **Filtering** should be set to /None or you will encounter communication problems.*

Legal values: /InterpreterBased, /None

Errors: **configurationerror**, **rangecheck**, **typecheck**

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

PrintHost string This parameter is a list of at most two IP mask/address pairs where the mask is applied to the given IP address to specify which hosts are allowed to make LPR connections. The slash is used as a delimiter between the two subfields. If two address pairs are specified, they are separated by a space delimiter. The mask, which has the same syntax as the IP address, is optional. If an address is specified with no corresponding mask, a mask of 255.255.255.255 is assumed.

A mask/address pair in which the mask is not specified, for example, would be:

138.46.24.37

Two IP mask/address pairs would have the format:

255.255.255.0/138.46.24.37 255.255.255.0/138.46.24.38

Legal values: An empty string or a string (of 63 or fewer non-null characters) which specifies up to 2 IP mask/addresses separated by the ASCII blank character. An IP address can be of the form *N.N.N.N* where each *N* is a decimal number in the range 0 to 255. IP addresses cannot be set to illegal values (e.g., trying to use an IP address equal to 0.0.0.0, 127.0.0.0, 255.255.255.255, *N.N.N*.255 or other illegal values will result in a **rangecheck** error).

Errors: **typecheck**, **limitcheck**, **rangecheck**

ReceiveWindowSize

integer Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations. The receive window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%.

Legal value: An integer in the range from 1024 to 65535.

Errors: **typecheck**, **rangecheck**

SendWindowSize integer Specifying the send window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations. The send window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%.

Legal value: An integer in the range from 1024 to 65535.

Errors: **typecheck**, **rangecheck**

StatusPortNumber integer This parameter denotes a port used by the unit for the purpose of sending status information back to the host. When using TCP/IP, communications is via UDP, not the TCP transport layer. The suggested default value is 9101. Users may use another port number to avoid a conflict with another unit on the network already using 9101.

Legal value: A positive integer representing a port number not reserved by any of the standard services.

	Warning	*An error is not raised when a port number is specified that has been previously reserved for some other purpose.*
		Errors: **typecheck**, **rangecheck**, **configurationerror**
Type	name	(*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.
		Legal value: /Communications
		Errors: None.

Telnet

Telnet gives network users interactive access to and exclusive use of the PostScript interpreter (or emulator). Port 23 is used for Telnet. Telnet is a TCP/IP network service.

Table 3.14 *Parameters present in the %Telnet% communications parameter set*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66.
		Legal values: *true*, *false*
		Errors: None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
HasNames	boolean	(*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.
		Legal value: *false*
		Errors: None.
Interpreter	name	For the general definition of **Interpreter**, see Table 3.4 on page 66.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

The Novell Remote Printer

This is an application that is managed by a Novell print server and takes print jobs downloaded from a print server. A remote printer may be shared by many print servers.

Table 3.15 *Parameters present in the %RemotePrinter% communications parameter set*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**

Filtering name For the general definition of **Filtering**, see Table 3.4 on page 66. **Filtering** allows for transport of data not initially intended for this network protocol. For example, if the host environment thought it was transmitting data over a parallel interface, such as LPT of a PC, and the printer is connected through a network interface, **Filtering** would need to be set to /InterpreterBased.

Legal values: /InterpreterBased, /None

Errors: **configurationerror**, **rangecheck**, **typecheck**

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

The Novell Print Server

This is the application that communicates with Novell file servers to download print jobs from the print queues. A print server may communicate with multiple file servers and access multiple print queues.

Table 3.16 *Parameters present in the %PrintServer% communications parameter set*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
Filtering	name	For the general definition of **Filtering**, see Table 3.4 on page 66. **Filtering** allows for transport of data not initially intended for this network protocol. For example, if the host environment thought it was transmitting data over a parallel interface, such as LPT of a PC, and the printer is connected through a network interface, **Filtering** would need to be set to /InterpreterBased. **Legal values:** /InterpreterBased, /None **Errors:** **configurationerror**, **rangecheck**, **typecheck**
HasNames	boolean	(*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66. **Legal value:** *false* **Errors:** None.
Interpreter	name	For the general definition of **Interpreter**, see Table 3.4 on page 66. **Legal values:** /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors:** **configurationerror**, **rangecheck**, **typecheck**
LoginPassword	string	This string parameter specifies the password that the print server application uses to gain access to the job queue. Setting this parameter to the empty string indicates that no password has been specified. The value of this

parameter returned by the **currentdevparams** operator is the string (INVALID) regardless of what the password is set to. Attempts to set the **LoginPassword** to the string (INVALID) will be ignored.

Legal values: A string of up to 32 characters.

Errors: **limitcheck**, **typecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PreferredServer string (*Read-write*) **PreferredServer** is the name of the fileserver that the PrintServer attempts to attach to in order to service queues. The validity of the **PreferredServer** name is not checked; the value is passed directly to the nearest fileserver for routing request. Novell 3.x fileserver names are limited to 47 characters; spaces and the characters " * + , \ / | ; : = < > ? [ ] are illegal. Novel 4.x limits the parameter length to 64 characters. Spaces are legal but not desirable. Novell 4.x converts spaces to underscores.

Note *We do not convert spaces, and spaces may not work on Novell 3.x.*

Legal values: A string of up to 64 characters. The default is an empty string.

Errors: **typecheck, limitcheck**

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

Type (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

Parameter Sets that Correspond to Various Network Services

This section covers the device parameter sets of type /Parameters that control various network services. The services are known as SNMP and Syslog.

Local Area Transport

The Local Area Transport (LAT) protocol is used primarily for communication between hosts and terminal servers. Often, these terminal servers have a printing device attached, making it possible to print from a given set of hosts to a particular set of printers on a local area network. The parameters in Table 3.17 on page 100 describe values germaine to the LAT protocol when used inside a printing device.

Table 3.17 *Parameters present in the %LAT% communications parameter set*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the general definition of **DelayedOutputClose**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** None.
Enabled	boolean	For the general definition of **Enabled**, see Table 3.4 on page 66. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
Filtering	name	For the general definition of **Filtering**, see Table 3.4 on page 66. **Filtering** allows for transport of data not initially intended for this network protocol. For example, if the host environment thought it was transmitting data over a parallel interface, such as LPT of a PC, and the printer is connected through a network interface, **Filtering** would need to be set to /InterpreterBased. **Legal values:** /InterpreterBased, /None **Errors:** **configurationerror**, **rangecheck**, **typecheck**
Groups	string	This parameter defines the groups allowed to access a device (printer device). A group is defined as an integer value in the range 0-255. Designate multiple groups in a string using a space as a delimiter; designate a range of groups using a dash. For example, (3-8 200 145-160). The default value is 0, which gives all groups access. **Legal values:** 0 to 255 **Errors:** **rangecheck**

HasNames boolean (*Read-only*) This parameter always has a value of *false*. For the general definition of **HasNames**, see Table 3.4 on page 66.

Legal value: *false*

Errors: None.

Interpreter name For the general definition of **Interpreter**, see Table 3.4 on page 66.

Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL

Errors: **configurationerror**, **rangecheck**, **typecheck**

KeepaliveTimer integer This parameter specifies the interval at which the circuit layer exchanges "keepalive" messages to maintain circuits when no session traffic is present.

Legal values: 10 to 180 seconds

Errors: **rangecheck**

MulticastTimer integer This parameter specifies the interval at which directory service advertisements (SAs) are multicast to advertise the printer's service.

Legal values: 10 to 1 80 seconds

Errors: **rangecheck**

On boolean For the general definition of **On**, see Table 3.4 on page 66.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Physical string (*Read-only*) This parameter specifies the physical layer over which LAT is accessed. The string is set to a device parameter set corresponding to a physical communications medium, such as the string (%EthernetPhysical%). A network layer parameter set can be associated with one and only one physical layer parameter set by the **Physical** parameter.

Legal values: A string of 32 or fewer non-null characters which specifies a physical layer.

Errors: None.

PrinterControl name For the general definition of **PrinterControl**, see Table 3.4 on page 66.

Legal value: /PSPrinter, /PJL

Errors: **configurationerror**, **rangecheck**, **typecheck**

RetransmitTimer integer This parameter specifies the interval at which the circuit layer retransmits unacknowledged messages.

Legal values: 1 to 10 seconds

Errors: **rangecheck**

RetransmitLimit integer This parameter specifies the number of retransmissions that will be attempted before a circuit is declared dead.

Legal values: 4 to 120

Errors: **rangecheck**

Type name (*Read-only*) This parameter always has a value of /Communications. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Communications

Errors: None.

SNMP

SNMP (Simple Network Management Protocol) provides a means for the system administrator to query for information about the unit. The information that can be queried is driven by a database called a Management Information Base (MIB). Refer to "A Simple Network Management Protocol" for details about SNMP[4]. It is not a communications port for PostScript language jobs, thus the parameter set is of type /Parameters. The parameters listed in Table 3.18 on page 103 are those SNMP parameters that need to be accessible from the PostScript language. These are the only parameters that are changeable from an environment separate from SNMP (the net-work side). The rules about when changes take effect to each parameter within this parameter set are described in Table 3.18 on page 103.

---

4. Case, Fedor, Schoffstall, and Davin, "A Simple Network Management Protocol," Request for Comments 1157, DDN Network Information Center, SRI International, May 1990.

Table 3.18 *Parameters present in the %SNMP% parameter set*

Key	*Type*	*Semantics*
PrivateHost	string	This parameter is a single node address (refer to Table 3.11 on page 88) per protocol of a host that is able to set those SNMP variables that can be written; an empty string indicates that no host has access. The empty string is the usual default value so that the unit will need to have this parameter explicitly set via the PostScript operator **setdevparams** prior to using SNMP. **Legal values:** An empty string or a string (of 49 or fewer non-null characters) which specifies up to one node address per protocol separated by the ASCII blank character. **Errors:** **typecheck**, **rangecheck, limitcheck**
SysContact	string	The convention is to use the name and phone number or address of the person responsible for the unit. Changes to this parameter take effect immediately. **Legal value:** A string of 32 or fewer non-null characters. **Errors:** **typecheck**, **limitcheck**
SysLocation	string	Location of the raster output device unit. Changes to this parameter take effect immediately. **Legal value:** A string of 32 or fewer non-null characters. **Errors:** **typecheck**, **limitcheck**
SysName	string	Name of the raster output device unit (expected by SNMP). Changes to this parameter take effect immediately. **Legal value:** A string of 32 or fewer non-null characters. **Errors:** **typecheck**, **limitcheck**
TrapHost	string	This parameter is a list of one or more (*node-address*/*community*) pairs for each protocol with a host that is able to receive traps. Refer to Table 3.11 on page 88 for the syntax of a node address. A slash is used as a delimiter between the *node-address* and the *community* string. The ASCII blank is used to separate each pair in the list. The *community* string portion is case insensitive. An empty string indicates that no traps are being sent to the host. Here are some example *community* strings: • public • proxy • private

- regional
- core

For example, the value (130.248.224.46/public) is an IP address for a trap host node in a public community.

The empty string is the usual default value so that the unit will need to have this parameter explicitly set via the PostScript operator **setdevparams** prior to using the trap host facility.

Legal values: An empty string or a string which specifies one or more (*node-address*/*community*) pairs separated by the ASCII blank character.

Errors: **typecheck**, **rangecheck**, **limitcheck**

Type name (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

Syslog

Syslog is a logging facility that sends log messages back to a UNIX host. The **LogPriority** value indicates which log messages will be seen by the host. Most of the messages contain network-specific information, but may include any other pertinent information the unit wishes to convey. Communication for %Syslog% is via the UDP (User Datagram Protocol) transport layer. Changes to this parameter set do not take effect until the unit is reinitialized.

Table 3.19 *Parameters present in the %Syslog% parameter set*

Key	*Type*	*Semantics*
LogHost	string	This string contains an IP address for a host that receives Syslog messages from the unit. An empty string indicates that no Syslog messages are to be sent by the unit. A null string implies that Syslog messages are disabled. **Legal values:** An empty string or a string (of 15 or fewer non-null characters) which specifies a legal IP address. An IP address is of the form *N.N.N.N* where each *N* is a decimal number in the range 0 to 255. Trying to use an IP address equal to 0.0.0.0, 255.255.255.255 or *N.N.N.*255 will result in a **rangecheck** error. **Errors:** **typecheck**, **limitcheck**

LogPriority integer This parameter designates which logging messages are to be sent on to the Syslog host. All logging messages associated with the specified **LogPriority** and those of higher priority (smaller numbers are higher priority) are sent. The following is a list, from the BSD (Berkeley Software Distribution) UNIX and SunOS™ convention, of priorities and their corresponding meaning.

0	Unit is no longer usable.
1	Messages indicating immediate action is needed on the part of a system administrator.
2	Critical error messages.
3	Error messages.
4	Warning messages.
5	Normal but significant conditions.
6	Informational messages.
7	Debugging messages.

Legal values: An integer in the range from 0 to 7.

Errors: **typecheck**, **rangecheck**

Type name (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

Communications Parameter Sets Which Correspond to Lower Protocol Layers

This section describes the device parameter sets of type /Parameters that control the transport, network, data link and physical layers of the TCP/IP or IPX/SPX protocol services.

TCP

TCP stands for Transmission Control Protocol and is the transport layer responsible for reliable data transfer by guaranteeing message delivery and reception. It is a connection-oriented protocol. If a packet is lost, it will be retransmitted. Changes to parameters in the TCP set do not take effect until the unit is reinitialized.

Table 3.20 *Parameters present in the %TCP% parameter set*

Key	*Type*	*Semantics*
On	boolean	A value of *true* means that the TCP protocol is activated at boot time. Otherwise it is off. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
ReceiveWindowSize	integer	Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations. **Legal values:** An integer in the range from 1024 to 65535. **Errors:** **typecheck**, **rangecheck**
SendWindowSize	integer	Specifying the send window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations. **Legal values:** An integer in the range from 1024 to 65535. **Errors:** **typecheck**, **rangecheck**
Type	name	(*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59. **Legal value:** /Parameters **Errors:** None.

User Datagram Protocol (UDP)

UDP is a connectionless (or datagram) protocol used in the TCP/IP networking suite. When using UDP with a peer host, there is no need for "handshaking" prior to communication. UDP packets are sent without any guarantee of delivery and may arrive at the destination in any order.

Table 3.21 *Parameters present in the %UDP% parameter set*

Key	*Type*	*Semantics*
Checksum	boolean	This boolean specifies whether checksum values will be inserted in outgoing packets formed by the software. The default should be *true*. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
On	boolean	A value of *true* means that the UDP protocol is activated at boot time. Otherwise it is off. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**
Type	name	(*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59. **Legal value:** /Parameters **Errors:** None.

IP

IP stands for Internet Protocol and is the network layer responsible for routing messages to their destinations. This layer decides which physical interface is to send outgoing messages and which transport layer is to receive incoming message. Changes to parameters in the IP set do not take effect until the unit is reinitialized.

Table 3.22 *Parameters present in the %IP% parameter set*

Key	*Type*	*Semantics*
BroadcastAddress	string	This parameter is the broadcast address mask used when broadcasting messages to the local network. **BroadcastAddress** shall reflect the current broadcast address mask in use by the unit. In order to “set” the **BroadcastAddress** explicitly and have it take effect the next time the unit is initialized, you must have **IPAddressDynamic** set to false when issuing setdevparams. *Note* *If the **BroadcastAddress** is not legal with respect to the **IPAddress** and **NetworkMask** given, it shall be changed to a value that is legal with no warning to the user. For example, suppose the **IPAddress** is 134.14.15.16 and the **BroadcastAddress** is 134.14.255.255. If the user changes*

IPAddress *to 134.15.15.16 without explicitly changing the* ***BroadcastAddress****, the* ***BroadcastAddress*** *shall automatically be changed to 134.15.255.255.*

Legal values: A string of 15 or fewer non-null characters that specifies a legal **BroadcastAddress**. A **BroadcastAddress** is of the form *N.N.N.N* where each *N* is a decimal number in the range 0 to 255.

Errors: **limitcheck**, **typecheck**, **rangecheck**

GatewayAddress string This parameter contains the *(destination-address/gateway-address)* pairs to other networks. The empty string specifies that dynamic routing, if available in the product, shall be enabled. In accordance with the route command, a **GatewayAddress** is fully defined here as a *destination-address* and a *gateway-address*. The slash is used as a delimiter between the two subfields. Multiple address pairs can be specified and are separated by a space delimiter. The number of *(destination-address/gateway-address)* pairs is implementation-dependent. In accordance with the route command, a *destination-address* is defined as specifying the network address. A valid network address varies according to the class. A class A network address requires the first field be non-zero. The others may be zero (for a net with no subnets), or contain subnet address information. A class B network address requires the first two fields to be non-zero. A class C network address requires the first three fields to be non-zero. A network address of 0.0.0.0 is a special case used by default if no previous network address matches the desired target IP address. If multiple entries have the same address, then the earlier entry will be ignored. **GatewayAddress** shall reflect the current (*destination address/gateway-address*) pairs to other networks in use by the unit. The **GatewayAddress** parameter may be “set” explicitly at any time. It will only take effect upon unit initialization, and then only if the **IPAddressDynamic** parameter is set to false via **setdevparams**. If **IPAddressDynamic** is set to true at unit initialization time, the **GatewayAddress** parameter will not take effect. The routing information will be gathered via dynamic routing using a RIP (Routing Information Protocol) request to the network. The default should be the empty string, implying dynamic routing.

Legal values: An empty string or a string which specifies one or more legal (*destination-address*/*gateway address*) pairs. The maximum length of the string is 32*n* - 1, where *n* is the number of *(destination-address/gateway-address)* pairs. The value *n* is product-specific. These addresses are Internet Protocol addresses of the form *N.N.N.N* where each *N* is a decimal number in the range

0 to 255. Loopback addresses (127.*N.N.N*) and broadcast addresses (*N.N.N*.255) are illegal for either the destination or gateway part of the pair.

Errors: **limitcheck**, **typecheck**, **rangecheck**

IPAddress string This is a unique string which represents the Internet Protocol address of the unit. The Internet Protocol address is mapped directly to the lowest physical address by which the unit is known (for example, **EthernetAddress** if **Physical** is %EthernetPhysical%). **IPAddress** shall reflect the current IP address in use by the unit. In order to "set" the **IPAddress** explicitly and have it take effect the next time the unit is initialized, you must have **IPAddressDynamic** set to *false* when issuing **setdevparams**. The default should be an empty string, which implies that the IP protocol layer is not active.

Note *Whenever **IPAddressDynamic** is true, **currentdevparams** will return a value for the parameter **IPAddress** that has been determined by a BOOTP or RARP sequence during boot up. Changing the **IPAddress** parameter to some other value via **setdevparams** has the effect of changing the user explicit value which is only used if **IPAddressDynamic** is false. **currentdevparams** will return the user explicit value of **IPAddress** only when **IPAddressDynamic** is false.*

Legal values: An empty string or a string (of 15 or fewer non-null characters) which specifies a legal IP address. An IP address is of the form *N.N.N.N* where each *N* is a decimal number in the range 0 to 255 (trying to set an IP address equal to 0.0.0.0, 255.255.255.255, 127.*N.N.N*, *N.N.N*.0, *N.N.N*.255 or any address whose first field is in the range 224 to 255 will result in a **rangecheck** error).

Errors: **limitcheck**, **typecheck**, **rangecheck**

IPAddressDynamic
boolean A value of true indicates that the **IPAddress** is obtained by a BOOTP or RARP (reverse address resolution protocol) sequence during boot up. The value of false means that the **IPAddress** must be explicitly set by a PostScript language job via **setdevparams** in order for connections to be made on the local network. The default value is usually false.

Legal values: *true*, *false*

Errors: **typecheck**

NetworkMask string This parameter indicates which fields of the **IPAddress** designate the network portion of the IP address and which designate the node portion. For example, the value 255.255.255.0 is a **NetworkMask** for a class B network

with subnets. The **NetworkMask** is used to determine if a certain IP address is on the same network as the unit. **NetworkMask** will reflect the current network mask in use by the unit. In order to "set" the **NetworkMask** explicitly and have it take effect the next time the unit is initialized, you must have **IPAddressDynamic** set to *false* when issuing **setdevparams**.

If the **IPAddress** is set to *true*, the **GatewayAddress** parameter should be set to appropriately legal values since dynamic routing is not always reliable when **IPAddress** is received via RARP.

Note *If **NetworkMask** is set to a value that is not legal when compared to the **IPAddress**, the **NetworkMask** will be changed to a value that is legal with no warning to the user. For example, if a class B **IPAddress** is given with a class A network mask, the **NetworkMask** shall be changed to the default class B network mask. The default class A network mask is 255.0.0.0. The default class B network mask is 255.255.0.0. The default class C network mask is 255.255.255.0. No subnets are accounted for in these default network masks.*

Legal values: A string of 15 or fewer non-null characters which specifies a legal IP mask. IP masks are of the form *N.N.N.N* where each *N* is a decimal number in the range 0 to 255.

Errors: **limitcheck**, **typecheck**

On boolean A value of *true* means that the IP protocol layer is activated at boot time. Otherwise it remains off.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Physical string (*Read-only*) This parameter specifies the physical layer over which IP is accessed. The string is set to a device parameter set corresponding to a physical communications medium, such as the string (%EthernetPhysical%). A network layer parameter set can be associated with one and only one physical layer parameter set by the **Physical** parameter.

Legal values: A string of 32 or fewer non-null characters which specifies a physical layer.

Errors: None.

TransmitEncapsulation

name This parameter specifies the transmit encapsulation type. /SNAP indicates either an 802.2 header or an 802.5 header with a SNAP header. /DIX indicates Ethernet II headers. The default value should be /DIX for Ethernet. The default value is (and can reasonably only be) /SNAP for TokenRing.

Legal values: /SNAP, /DIX

Errors: **typecheck**

Note *These new values have been introduced to eliminate dependencies on the type of connection (Ethernet or TokenRing) used. Legal values in the 2016 Supplement were /802.3-2-SNAP, /DIX and /802.5-2-SNAP.*

Type name (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

Sequenced Packet Exchange (SPX) Protocol

SPX is the Novell Netware connection oriented protocol. When using SPX to communicate with a peer host, there is “handshaking” before the connection is ready. The delivery of SPX packets are expected to be acknowledged to guarantee delivery and packets arrive in sequence at the destination. Unlike TCP, it does not provide a sliding window functionality for flow control.

Table 3.23 *Parameters present in the %SPX% parameter set*

Key	*Type*	*Semantics*
On	boolean	A value of *true* means that the SPX protocol is activated at boot time. Otherwise it is off. **Legal values:** *true*, *false* **Errors:** **configurationerror**, **typecheck**

ReceiveWindowSize

integer — Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations.

Legal values: An integer in the range from 1024 to 59392.

Errors: **typecheck**, **rangecheck**

Type — name — (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

Internetwork Packet Exchange (IPX) Protocol

IPX is the Novell Netware connectionless (or datagram) protocol. When using IPX with a peer host, there is no need for "handshaking" prior to communication. IPX packets are sent without any guarantee of delivery and may arrive at the destination in any order. Netware broadcasting is done using IPX.

Table 3.24 *Parameters present in the %IPX% parameter set*

Key	*Type*	*Semantics*
Checksum	boolean	This boolean specifies whether checksum values will be inserted in outgoing packets formed by the software. The default should be *true*. **Legal values:** *true*, *false* **Errors:** None.
HopCount	integer	(*Read-write*) This parameter specifies the maximum number of routers the print server will go through in trying to attach to fileservers while looking for queues to service. The preferred value is the smallest value needed to reach all of the printer's servers. A count of 15 is defined as trying to reach all reachable servers; a count of 16 is defined as unreachable. If the **PreferredServer** parameter is set, **HopCount** is ignored unless the **PreferredServer** is unreachable. A negative value or a value larger than 15 will default to 15. **Legal values:** 0 through 15. **Errors:** **typecheck**

NetworkAddress string (*Read-only*) This parameter identifies the network in which the unit is located. The concatenation of the **NetworkAddress** and the Novell Node Number will uniquely identify the unit on the network. The Novell Node Number is derived from the Media Access Control (MAC) address of the networking media. For Ethernet, the Novell Node Number is the **EthernetAddress** parameter of the %EthernetPhysical% set. The **NetworkAddress** is obtained from the Novell file server on the local net upon booting the printer.

Legal values: An empty string or a string (of 8 or fewer non-null characters) which specifies a legal Novell network address. A Novell network address is of the form *XXXXXXXX* where each *X* represents a digit in hexadecimal in the range 0 to F.

Errors: None.

On boolean A value of *true* means that the IPX protocol layer is activated at boot time. Otherwise it remains off.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Physical string (*Read-only*) This parameter specifies the physical layer over which IPX is accessed. The string is set to a device parameter set corresponding to a physical communications medium, such as the string (%EthernetPhysical%). A network layer parameter set can be associated with one and only one physical layer parameter set by the **Physical** parameter.

Legal values: A string of 32 or fewer non-null characters which specifies a physical layer.

Errors: None.

TransmitEncapsulation

name This parameter specifies the transmit encapsulation type. /NO_SNAP indicates either an 802.2 or 802.5 header without a SNAP header. /SNAP indicates either an 802.2 or 802.5 header with a SNAP header. The default value should be /802.3 when **Physical** is %EthernetPhysical% and /NO_SNAP when **Physical** is %TokenRingPhysical%. /DIX and /802.3 are applicable only to Ethernet. The default value is 802.3. /DIX indicates Ethernet Version II. /Adaptive indicates that, by the nature of the interaction between host and printer, an encapsulation format to use in responses to the host can be derived at boot time.The value of this parameter is checked solely for legality; it is not checked for applicability.

The relationships between the values of **TransmitEncapsulation**, **Physical**, /EthernetPhysical, and /TokenRingPhysical are as follows:

Novell FrameType	*TransmitEncapsulation*	*Physical*
Ethernet_802.3	802.3	/EthernetPhysical
Ethernet_802.2	NO_SNAP	/EthernetPhysical
TOKEN_RING	NO_SNAP	/TokenRingPhysical
Ethernet_SNAP	SNAP	/EthernetPhysical
TOKEN_RING_SNAP	SNAP	/TokenRingPhysical
Ethernet_II	DIX	/EthernetPhysical
Any	Adaptive	/EthernetPhysical or /TokenRingPhysical

Legal values: /802.3, /NO_SNAP, /SNAP, /DIX, /Adaptivel

Errors: **typecheck**

Note *These new values have been introduced to eliminate dependencies on the type of connection used. Consult the 2016 Supplement for alternative values.*

Type name (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

EthernetPhysical

The %EthernetPhysical% device parameter set corresponds to a physical Ethernet connector and its associated hardware and the data link layer software which handles events from this device. Changes to parameters in this set do not take effect until the unit is reinitialized.

Table 3.25 *Parameters present in the %EthernetPhysical% parameter set*

Key	*Type*	*Semantics*
ConnectorType	name	(*Read-only*) This parameter indicates which Ethernet connector type is being used. **Legal values:** /RJ45, /BNC, /AUI, /AAUI **Errors:** None.
EthernetAddress	string	*(Read-only)* This parameter returns a unique string that represents the Ethernet address of the unit. The string is of the form (*XX*:*XX*:*XX*:*XX*:*XX*:*XX*), where each *X* represents a digit in hexadecimal.

Note *When using Novell, the Ethernet address is also known as the Novell Node Number.*

Legal values: A string of 17 characters representing a legal Ethernet address.

Errors: None.

Name string *(Read-only)* This parameter specifies the mnemonic name, such as (le0) for "Lance chip interface unit 0" or (so0) for "Sonic chip interface unit 0", for the Ethernet interface used.

Legal values: Any string of 16 or fewer non-null characters.

Errors: None.

On boolean A value of *true* means that the Ethernet channel is enabled at boot time. Otherwise it remains off.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Type name (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

TokenRingPhysical

The %TokenRingPhysical% device parameter set corresponds to a physical Token Ring connector and its associated hardware and the data link layer software which handles events from this device. Changes to parameters in this set do not take effect until the unit is reinitialized.

Table 3.26 *Parameters present in the %TokenRingPhysical% parameter set*

Key	*Type*	*Semantics*
Address	string	*(Read-only)* This parameter returns a unique string that represents the Ethernet address of the unit. The string is of the form (*XX*:*XX*:*XX*:*XX*:*XX*:*XX*), where each *X* represents a digit in hexadecimal.

Legal values: A string of 17 characters representing a legal token ring address.

Errors: None.

Bridging name Bridging, on the token ring, can be done in several different ways. When this parameter is set to /Transparent, this implies a transparent bridging where the entire “universe” is one large single ring structure and all identities are unique. When set to /SourceRoute, routing is done via specifying an explicit path including the ring identification, RIF. When set to /Adaptive, the software will automatically recognize the routing style and respond in kind (either as a one-time determination or when processing each connection).

Legal values: /Transparent, /SourceRoute, /Adaptive

Errors: **configurationerror**, **typecheck**

ConnectorType name *(Read-only)* This parameter indicates which TokenRing connector type is being used.

Legal values: /RJ45, /DB9, /MAU

Errors: None.

Name string *(Read-only)* This parameter specifies the mnemonic name, such as (le0) for “Lance chip interface unit 0” or (so0) for “Sonic chip interface unit 0”, for the Token Ring interface used.

Legal values: Any string of 16 or fewer non-null characters.

Errors: None.

On boolean A value of *true* means that the Token Ring channel is enabled at boot time. Otherwise it remains off.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

Speed integer This parameter indicates the speed at which the ring is operated, in megabits per second.

Legal values: 4, 16

Errors: **configurationerror**, **typecheck**

Type name (*Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

3.5.3 File System Parameters

Parameters Present in Parameter Sets of Type /FileSystem

Table 3.27 lists the parameters common to device sets of type /FileSystem.

Table 3.27 *Parameters common to device sets of type /FileSystem*

Key	*Type*	*Semantics*
HasNames	boolean	*(Read-only)* This parameter indicates whether the device represented by the parameter set supports named files. If the device is not mounted, this parameter has a value of *false*. This is defined only in device parameter sets of the **Type** /FileSystem or /Communications. **Legal values:** *true*, *false* **Errors:** None.

Disk, Cartridge, ROM and RAM Parameter Tables

Table 3.28 on page 117 contains a list of the current disk parameters. Table 3.29 on page 122 contains a list of the current cartridge or ROM parameters. The name "%rom%" is used instead of "%cartridge%" for a cartridge that is non-removable and non-writeable. Table 3.30 on page 124 contains a list of the current RAM file system parameters. The name "%ram%" is used for a file system that is writeable and stored in some form of RAM.

Read-only refers to their access by language operators (for example, **setdevparams**, **currentdevparams**). A *read-only* parameter can change value but not as the result of invoking **setdevparams**. Changes to parameters of type /FileSystem take place immediately.

In Table 3.28 on page 117, Table 3.29 on page 122, and Table 3.30 on page 124, it should be understood that a page is a unit of storage whose size is file-system-dependent.

Table 3.28 *Parameters present in %disk% (/FileSystem) devices*

Key	*Type*	*Semantics*
BlockSize	integer	*(Read-only)* This parameter indicates the disk/cartridge formatting size of a page (for the logical and physical size of the media). The formatting size of a page for a cartridge is 1 byte per block. The formatting size of a page for a disk using the Adobe file system is 1024 bytes per block. **Legal values:** A positive integer, typically 1024. **Errors:** None.

Bus string *(Read-only*) With the Adobe storage device implementation, **Bus** will indicate the name of the bus on which this disk resides. This parameter is in the form of a string that can be used as input to **setdevparams** or **currentdevparams** to get bus parameters.

Legal values: %Scsi%, %ScsiB%, %ScsiC%, %Ide%, %IdeB%, %IdeC%, and so forth.

Errors: None.

Free integer *(Read-only*) This parameter indicates the amount of free space available on the media for the device in *pages*, where the *page* size is indicated by the parameter **BlockSize**. This parameter is valid only if the device is mounted (that is, **Mounted** is set to *true*). A value of 0 indicates that either the device is not mounted or the media is completely full.

Legal values: 0 or any positive integer.

Errors: None.

HasNames boolean *(Read-only)* This parameter indicates whether the device supports named files. This parameter is valid only if the device is mounted (that is, **Mounted** is set to *true*). If the device is not mounted, this parameter has a value of *false*.

Legal values: *true, false*

Errors: None.

InitializeAction integer This parameter specifies an action for initializing the device. The following are valid values for disks:

0	Indicates no action and is the value returned when the parameter is read.
1	Indicates that the current file system (if any) is to be deleted and a new one of size **LogicalSize** created (the media is assumed to have been formatted already). The device must first be mounted; otherwise an **ioerror** will result. For more information, see **LogicalSize**.
2	Reformats the entire media before creating a new file system of size **LogicalSize**. The **Interleave** parameter also plays a role in how the media is to be formatted. See **Interleave** below for details.

3 Or greater has the same effect as the value 2 and also carries out product-dependent actions, which typically consist of reformatting the disk and running integrity tests before creating the file system. Some devices can have additional parameters that serve as arguments to **InitializeAction**.

Legal values: Any non-negative integer.

Errors: **ioerror**

Interleave integer The purpose of **Interleave** is to arrange logically contiguous sectors on the disk in a way that is most efficient for the system using that disk. This parameter is used only when the media is being formatted (refer to **InitializeAction**, described above).

For example, assume there are 16 sectors going around a single track on a disk. If the first sector has a logical number of 1, the second 2, the third 3 and so on, it is referred to as “1 to 1 interleave” and the value of **Interleave** is 1. In this case the system must be very fast in order to be able to take data from the disk, one sector immediately after another. If the system fails to consume the first sector in time for the second sector, the system has to wait an entire revolution of the disk to get the next sector. This can give very poor performance.

If the first sector has a logical number of 1, the third has a logical number of 2, the fifth has a logical number of 3 and so on, the system will need to be able to consume the current sector while the head passes over a sector in time for the next logical sector. This is referred to as “2 to 1 interleave” and the value of **Interleave** is 2. The sectors in between are used for higher logical numbers and it takes a minimum of two revolutions to get an entire track’s data off the disk. In this example, the second physical sector on the disk would be between logical sectors 1 and 2 and it would be logical number 9.

Similarly, “3 to 1 interleave” has an **Interleave** value of 3 and the first sector has a logical number of 1, the fourth one of 2 and so on. Normally, the interleave should be set to a value that allows the software to use the information during the time between sectors, but not waste any time. It is difficult to determine what the proper value is and it is highly dependent on the job accessing the disk. Some drives provide buffering for a full track of data. For these drives, “1 to 1 interleave” is almost always most efficient.

Legal values: Any positive integer; the legality of the value is disk-dependent.

Errors: **ioerror**

LogicalSize integer When set, this parameter specifies the size of the file system to be created and is used as an argument to the action carried out by **InitializeAction**. If **LogicalSize** is 0, **InitializeAction** uses a default size that is normally the size of the entire media within the device. For more information, see **InitializeAction**.

When queried, this parameter indicates the *current* size of the file system on the device in *pages*, where the *page* size is indicated by the parameter **BlockSize**. A value of 0 indicates that the device is not mounted.

If **LogicalSize** is set with a certain value and then the device is reformatted, a query of **LogicalSize** should return the value that was set. However, if the parameter is queried at any time before the media within the device is reformatted, it may return a different value from what was set because it may return the *current* size.

Legal values: Any non-negative integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**.

Errors: **rangecheck**, **typecheck**

Mounted boolean If this parameter is set to *true*, the system attempts to mount the device. If set to *false*, the system attempts to dismount the device. Mounting a device makes it known to the system and makes it at least readable, depending on the nature of the device. A device will not mount successfully if it does not contain a valid file system.

When queried, the return value indicates whether the device is *currently* mounted. Obtain the result of an attempted mount by querying **Mounted** immediately after setting it.

Mounted raises a **configurationerror** if it is set to *true* and mounting fails, or if it is set to *false* and dismounting fails.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck, ioerror**

PhysicalSize integer *(Read-only)* This parameter indicates the size of the media in *pages*, where the *page* size is indicated by the parameter **BlockSize**. This parameter is only valid when the device is mounted (that is, **Mounted** is set to *true*). A value of 0 indicates that the device is not mounted.

Legal values: Any non-negative integer or 0.

Errors: None.

PrepareAction integer This parameter specifies an action to prepare the underlying filesystem for a specific purpose. Valid values are:

0	Indicates no action is to be performed (no-op).
1	Causes a product-specific action to load system files. In one case, these files support older versions of Adobe Japanese typefaces. On a writable file system, these system files enable older versions of the Japanese Font Downloader utility to work correctly.

Note *If **InitializeAction** and **PrepareAction** are set in the same invocation of **setdevparams**, the actions performed by **InitializeAction** precede those performed by **PrepareAction**.*

Legal values: 0, 1

Errors: **rangecheck**, **ioerror**

Removable boolean *(Read-only)* This parameter indicates whether the device supports removable media. Depending on how the removable media device operates, setting **Mounted** to *false* will either eject the media or allow its removal. When the media has been removed, it cannot be mounted again until it is re-inserted.

Legal values: *true*, *false*

Errors: None.

Searchable boolean This parameter indicates whether the device participates in searches in file system operations that have specified a file name without specifying a device. See section 3.8.2, “Named Files,” of the *PostScript Language Reference Manual, Second Edition* for more information.

Note *Devices that support removable media (on some products) will initially have **Searchable** set to false. **Searchable** must be explicitly set to true to have the media be searched.*

Legal values: *true*, *false*

Errors: None.

SearchOrder integer This parameter indicates the priority at which the device participates when searching for a file in operations where no device has been specified. A lower integer indicates a higher priority. This parameter is ignored if the **Searchable** parameter is *false*.

Legal values: Any non-negative integer.

Errors: None.

Type name *(Read-only)* This parameter always has a value of /FileSystem. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /FileSystem

Errors: None.

Writeable boolean *(Writeable, but only during a mount)* This parameter indicates whether the files on the device can be open for write access. This parameter can be set to *true* or *false* only during a mount (that is, when **Mounted** is being set to *true* in a call to **setdevparams**) and only if the media is not write-protected. If the media is already write-protected, this parameter is a constant equal to *false*. When the device is not mounted, this parameter indicates whether or not the drive will support writeable media.

Legal values: *true*, *false*

Errors: None.

Table 3.29 lists the parameters present in %cartridge% and %rom% (/FileSystem) devices.

Table 3.29 *Parameters present in %cartridge% or %rom% (/FileSystem) devices*

Key	*Type*	*Semantics*
BlockSize	integer	*(Read-only)* For the general definition of **BlockSize**, see Table 3.28 on page 117. **Legal values:** Any non-zero positive integer (typically 1). **Errors:** None.
CartridgeID	integer	*(Read-only)* This parameter indicates an ID that uniquely identifies this cartridge on a product. **CartridgeID** is used by the interpreter to determine if a cartridge has been removed from a slot and a different cartridge inserted. **Legal values:** Any integer. **Errors:** None.
CartridgeType	integer	*(Read-only)* This parameter indicates the category classification of the cartridge. This classification is a registry maintained by Adobe. **Legal values:** Any integer. **Errors:** None.

Free integer *(Read-only*) For the general definition of **Free**, see Table 3.28 on page 117.

Legal values: Any non-negative integer or 0.

Errors: None.

HasNames boolean *(Read-only)* For the general definition of **HasNames**, see Table 3.27 on page 117.

Legal values: *true*, *false*

Errors: None.

InitializeAction integer This parameter specifies an action for initializing the device. The following are valid values for writeable cartridges (setting **InitializeAction** for a read only cartridge has no effect):

0	Indicates no action and is the value returned when the parameter is read.
1	Reformats the entire media and then creates a new file system using the full size of the cartridge.

Legal values: 0 or 1.

Errors: **ioerror**

LogicalSize integer For the general definition of **LogicalSize**, see Table 3.28 on page 117.

Legal values: Any non-negative integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**.

Errors: **rangecheck**, **typecheck**

Mounted boolean For the general definition of **Mounted**, see Table 3.28 on page 117.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PhysicalSize integer *(Read-only*) For the general definition of **PhysicalSize**, see Table 3.28 on page 117.

Legal values: Any non-negative integer or 0.

Errors: None.

Removable boolean *(Read-only)* For the general definition of **Removable**, see Table 3.28 on page 117.

Legal values: *true*, *false*

Errors: None.

Searchable boolean For the general definition of **Searchable**, see Table 3.28 on page 117.

Legal values: *true*, *false*

Errors: None.

SearchOrder integer For the general definition of **SearchOrder**, see Table 3.28 on page 117.

Legal values: Any non-negative integer.

Errors: None.

Type name *(Read-only)* This parameter always has a value of /FileSystem. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /FileSystem

Errors: None.

Writeable boolean *(Writeable, but only during a mount)* For the general definition of **Writeable**, see Table 3.28.

Legal values: *true*, *false*

Errors: None.

Table 3.30 lists the parameters present in %ram% (/FileSystem) devices.

Table 3.30 *Parameters present in %ram% (/FileSystem) devices*

Key	*Type*	*Semantics*
BlockSize	integer	*(Read-only)* For the general definition of **BlockSize**, see Table 3.28 on page 117. **Legal values:** Any non-zero positive integer (typically 1). **Errors:** None.

Free integer *(Read-only*) For the general definition of **Free**, see Table 3.28 on page 117.

Legal values: Any non-negative integer or 0.

Errors: None.

HasNames boolean *(Read-only)* For the general definition of **HasNames**, see Table 3.27 on page 117.

Legal values: *true*, *false*

Errors: None.

InitializeAction integer This parameter specifies an action for initializing the device. The following are valid values for writeable file systems in RAM:

0	Indicates no action and is the value returned when the parameter is read.
1	Reformats the entire media and then creates a new file system using the full size of the cartridge.

Legal values: 0 or 1.

Errors: **ioerror**

LogicalSize integer For the general definition of **LogicalSize**, see Table 3.28 on page 117. The setting of **LogicalSize** when **InitializeAction** is processed designates the amount of memory the user wants to allocate to the %ram% file system. Actual allocation may be less and may not exceed the value of **PhysicalSize**.

Legal values: Any non-negative integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**.

Errors: **rangecheck**, **typecheck**

Mounted boolean For the general definition of **Mounted**, see Table 3.28 on page 117.

Legal values: *true*, *false*

Errors: **configurationerror**, **typecheck**

PhysicalSize integer *(Read-only*) For the general definition of **PhysicalSize**, see Table 3.28 on page 117. The value of **PhysicalSize** is set to the maximum allowable size of the %ram% file system. The user may designate a smaller allocation using the **LogicalSize** parameter.

Legal values: Any non-negative integer or 0.

Errors: None.

Searchable boolean For the general definition of **Searchable**, see Table 3.28 on page 117.

Legal values: *true*, *false*

Errors: None.

SearchOrder integer For the general definition of **SearchOrder**, see Table 3.28 on page 117.

Legal values: Any non-negative integer.

Errors: None.

Type name *(Read-only*) This parameter always has a value of /FileSystem. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /FileSystem

Errors: None.

Writeable boolean *(Writeable*, *but only during a mount*) For the general definition of **Writeable**, see Table 3.28.

Legal values: *true*, *false*

Errors: None.

3.5.4 %os% Device Parameters

The %os% device parameter set is present only in Display PostScript products or in other products which are UNIX-based. It is not present in printers. Since the UNIX file system is an entity separate from PostScript, most of the %os% parameter set has constant, read-only values. A minimal description set of parameters is provided primarily for consistency with other types of file systems. Table 3.31 lists the parameters present in the %os% parameter sets.

Table 3.31 *Parameters present in the %os% parameter sets*

Key	*Type*	*Semantics*
HasNames	boolean	(*Read-only*) This parameter always returns a value of *true*. For the general definition of **HasNames**, see Table 3.27 on page 117. **Legal value:** *true* **Errors:** None.

InitializeAction integer (*Read-only*) This parameter always returns a value of 0. For the general definition of **InitializeAction**, see Table 3.28 on page 117.

Legal value: 0

Errors: None.

Mounted boolean (*Read-only*) This parameter always returns a value of *true*. For the general definition of **Mounted**, see Table 3.28 on page 117.

Legal value: *true*

Errors: **typecheck**

Removable boolean (*Read-only*) This parameter always returns a value of *false*. For the general definition of **Removable**, see Table 3.28 on page 117.

Legal value: *false*

Errors: None.

Searchable boolean (*Read-only*) This parameter always returns a value of *true*. For the general definition of **Searchable**, see Table 3.28 on page 117.

Legal values: *true*, *false*

Errors: None.

SearchOrder integer This parameter *initially* returns a value of 2. For the general definition of **SearchOrder**, see Table 3.28 on page 117.

Legal values: Any non-negative integer.

Errors: None.

Type name (*Read-only*) This parameter always returns a value of /FileSystem. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /FileSystem

Errors: None.

Writeable boolean (*Read-only*) This parameter always returns a value of *true*. For the general definition of **Writeable**, see Table 3.28 on page 117.

Legal value: *true*

Errors: None.

3.5.5 SCSI Bus Parameter Set

The parameters in Table 3.32 on page 128 are used to configure the SCSI bus. The (%Scsi%) parameter set is always present in a printer that has a SCSI bus, even if no devices are present on the SCSI bus. If more than one SCSI bus is present, the first is called %Scsi%, the second %ScsiB%, the third %ScsiC% and so on. Changes to SCSI parameters do not take effect until the next time the system is initialized.

Table 3.32 *Parameters present in the %Scsi% parameter sets*

Key	*Type*	*Semantics*
BootDelay†	integer	This parameter indicates how long the disk I/O driver should wait (in seconds) during system initialization for the disk to spin up, before determining that a disk is not present or not responding. A value of 0 means that there is no waiting for the disk to spin up. You should set this parameter in accordance with the characteristics of the disk attached to the printer. **Legal values:** 0 or any positive integer. **Errors:** None.
CheckParity	boolean	This parameter indicates if parity on the SCSI bus is to be checked. The default value is usually *true*. *Warning* — *Setting **CheckParity** to true on products that do not support parity checking would be unwise. Refer to the individual product addendum for information on whether or not a given product can do this checking.* **Legal values:** *true, false* **Errors:** None.
InitiatorId†	integer	This parameter is the address on the SCSI bus used by the printer when it serves as initiator. The default value is usually 7. **Legal values:** Any integer in the range 0 to 7. **Errors:** **configurationerror**
Poll†	integer	This parameter is a bit-encoded specification of which addresses on the SCSI bus should be polled by the printer when it looks for disks during system initialization. For example, a 1 in bit 0 means poll for %disk0%. Any bits in this mask which correspond to addresses that are used as the printer's **InitiatorId** or **TargetId**, as the **InitiatorId** for other hosts on the bus, or as the **TargetId** of peripherals belonging to other hosts on the bus should be set to 0 (meaning "do not poll"). If the bit is set to poll the address corresponding to the printer's **InitiatorId** or **Target Id**, a **configurationerror** is generated. If the

bit is set to poll an address that shouldn't be polled, anomalies may occur on the bus. **Poll** is expressed as an integer bit mask in the range 0 to 254 (never 255 since all bits cannot be on—one bit must be reserved for **InitiatorId**). The default value is usually 127 (7F in hexadecimal).

Legal values: An integer bit mask in the range 0 to 254.

Errors: **configurationerror**

TargetId* integer This parameter is the SCSI bus address reserved by the printer for use as the %ScsiComm% communications channel. This address may be the same as the **InitiatorId**.

Legal values: An integer in the range 0 to 7.

Errors: **configurationerror**

Type name (*Read only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

* = present only when ScsiComm is used on the bus

† = present only when disks are present on the bus

3.5.6 IDE Bus Parameter Set

The parameters in Table 3.33 on page 130 are used to configure the IDE bus. The (%Ide%) parameter set is always present in a printer that has an IDE bus, even if no devices are present on the IDE bus. If more than one IDE bus is present, the first is called %Ide%, the second %IdeB%, the third %IdeC% and so on. Changes to IDE parameters do not take effect until the next time the system is initialized.

Table 3.33 *Parameters present in the %Ide% parameter sets*

Key	*Type*	*Semantics*
BootDelay†	integer	This parameter indicates how long the disk I/O driver should wait (in seconds) during system initialization for the disk to spin up, before determining that a disk is not present or not responding. A value of 0 means that there is no waiting for the disk to spin up. You should set this parameter in accordance with the characteristics of the disk attached to the printer. **Legal values:** 0 or any positive integer. **Errors:** None.
Poll†	integer	This parameter is a bit-encoded specification of which addresses on the IDE bus should be polled by the printer when it looks for disks during system initialization. For example, a 1 in bit 0 means poll for %disk0%. **Poll** is expressed as an integer bit mask in the range 0 to 3. **Legal values:** An integer bit mask in the range 0 to 3. **Errors:** **configurationerror**
Type	name	(*Read only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59. **Legal value:** /Parameters **Errors:** None. † = present only when disks are present on the bus

3.5.7 Engine Device Parameters

Table 3.34 on page 130 lists the parameters associated with the engine device.

Table 3.34 *Parameters present in the %Engine% parameter set*

Key	*Type*	*Semantics*
BSizeStandard	name	This parameter assists the engine in determining the physical dimensions of the paper when B4 or B5 paper is selected. There are two choices for the value of **BSizeStandard**:

/ISO: ISO is the abbreviation for the International Standards Organization, which is the international body that defines the “metric” paper sizes (A4, A3, B5, B4, B3 and so forth). These are the paper sizes used in Europe and much of the rest of the world. The table below lists the dimensions for the B4 and B5 paper sizes as defined by ISO:

Paper Size	*Dimensions*
B4	250 x 353 mm. or 709 x 1001 default units.
B5	176 x 250 mm. or 499 x 709 default units.

/JIS: JIS is the abbreviation for the Japanese Institute for Standardization, the national body which specifies standards for use in the country of Japan. Japan also uses the standard “A” paper sizes. However, they use a slightly different definition of the “B” paper sizes. The table below lists the dimensions for the B4 and B5 paper sizes for JIS:

Paper Size	*Dimensions*
B4	257 x 364 mm. or 729 x 1032 default units.
B5	182 x 257 mm. or 516 x 729 default units.

Note *In the above tables, a “default unit” denotes 1/72 of an inch.*

Legal values: /ISO, /JIS

Errors: **rangecheck**

Darkness real This parameter controls the overall lightness or darkness of the rendered page on a monochrome device. This parameter does not affect the frame buffer, nor does it have any computational overhead. Legal values are real numbers from 0.0 through 1.0. A value of 0.0 means minimum darkness, 1.0 means maximum darkness. This option is provided in some products whose marking hardware allows software control of colorant application. The default value is product-dependent.

Legal values: Real numbers in the range 0.0 to 1.0, inclusive.

Errors: **rangecheck**

DarknessBlack real This parameter controls the overall lightness or darkness of the black color on a rendered page produced on a device with multiple toner stations. (See **Darkness** for a complete description.)

Legal values: Real numbers in the range 0.0 to 1.0 inclusive.

Errors: **rangecheck**

DarknessCyan real This parameter controls the overall lightness or darkness of the cyan color on a rendered page. (See **Darkness** for a complete description.)

Legal values: Real numbers in the range 0.0 to 1.0 inclusive.

Errors: **rangecheck**

DarknessMagenta real This parameter controls the overall lightness or darkness of the magenta color on a rendered page. (See **Darkness** for a complete description.)

Legal values: Real numbers in the range 0.0 to 1.0 inclusive.

Errors: **rangecheck**

DarknessYellow real This parameter controls the overall lightness or darkness of the yellow color on a rendered page. (See **Darkness** for a complete description.)

Legal values: Real numbers in the range 0.0 to 1.0 inclusive.

Errors: **rangecheck**

PageCount integer This parameter is a count of all pages fed by the engine. The count includes all of the pages successfully printed as well as the pages that were jammed or spoiled. The value of **PageCount** is determined by querying the engine.

Legal values: Any non-negative integer or 0.

Errors: **typecheck**

TimeToStandby integer After the specified number of minutes, the engine will go into a “standby” mode, in which it stops trying to keep itself ready to print a page; that is, it stops keeping its fuser hot. The next time the controller sends a feed or prefeed command, the engine will enter the “warming up” state until it is ready to print. The range of acceptable values for **TimetoStandby** are product-specific. An unallowed value is rounded to the nearest allowed value. Specifying a value of 0 for this parameter has the effect of never letting the printer enter the “standby” mode.

Legal values: Product-specific. Typically an integer in the range 0 – 720.

Errors: **rangecheck**

Type name *(Read-only*) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

3.5.8 Console Device Parameters

The %Console% device parameter set provides a means of setting and controlling characteristics of the operator console of an output device which includes the PostScript language. The keys currently defined in this set provide an extensible way of selecting the natural language (for example, English, Japanese) in which information will be displayed on the operator console. Table 3.35 on page 133 lists the parameters associated with the console device.

Table 3.35 *Parameters present in the %Console% parameter set*

Key	*Type*	*Semantics*
CharSet	name	This parameter specifies the name of a character set. This may be either the name of a standard character set or a vendor specific character set. If it is a standard character set, the name will designate the standard (and where applicable, the variant within the standard); for example, ISO-646-ISV (for ASCII). If it is vendor specific, then the name should designate the vendor and the identification of the character set used by that vendor; for example, IBM-Codepage-550. Because the same character set may be known by several names (for example, ASCII and ISO-646-ISV), aliases are allowed for character set names; that is, the same character set may be designated by more than one name. **Legal values:** /ASCII This is the same as ISO-646-IRV except for the “$.” /ISO-646-ISV This is ASCII with a currency symbol instead of “$.” /ISO-8859-1 This is the ISO 8-bit Latin-1 characters set. /Adobe-Japan1-0 This is the CID-keyed Japanese character collection. **Errors:** **rangecheck**
Country	name	This parameter indirectly specifies the dialect of the language by referring to the country in which the dialect is used. The country is indicated using a name which is a two character country code from the ISO 3166 Standard (these codes represent the names of countries). **Legal values:** (source - ISO 3166**)** *Value* *Meaning* /AR Argentina /AU Australia

/BE	Belgium
/BO	Bolivia
/BR	Brazil
/CA	Canada
/CL	Chile
/CN	Peoples Republic of China
/TW	Taiwan
/CO	Columbia
/CS	Czechoslovakia (probably obsolete)
/DK	Denmark
/EC	Ecuador
/FI	Finland
/FR	France
/DE	Germany
/GR	Greece
/HU	Hungary
/IN	India
/ID	Indonesia
/IE	Ireland
/IL	Israel
/IT	Italy
/JP	Japan
/LU	Luxembourg
/MX	Mexico
/NL	Netherlands (Holland)
/NZ	New Zealand
/NO	Norway
/PK	Pakistan
/PA	Panama
/PY	Paraguay
/PE	Peru
/PH	Philippines
/PL	Poland

/PT	Portugal
/SA	Saudi Arabia
/ZA	South Africa
/ES	Spain
/SE	Sweden
/GB	United Kingdom (England, Wales, Scotland, N. Ireland)
/US	United States
/SU	Soviet Union (probably obsolete)
/VE	Venezuela

Errors: **typecheck**

Language name This parameter specifies a name which is a two character language code from the ISO 639 Standard, Code for the representation of names of languages.

Legal values: (source - ISO 639)

Value	*Meaning*
/CS	Czech
/DA	Danish
/DE	German
/EL	Greek
/EN	English
/FI	Finnish
/FR	French
/GA	Irish
/HU	Hungarian
/IT	Italian
/IW	Hebrew
/JA	Japanese
/KO	Korean
/NL	Dutch
/NO	Norwegian
/PL	Polish
/PT	Portuguese
/RU	Russian
/SK	Slovak

/SV Swedish

/ZH Chinese

Errors: **rangecheck**

Type name *(Read-only)* This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

3.5.9 Emulator Parameters

An emulator is an alternative interpreter for the input stream. Some PostScript printers have the ability to emulate other printers. The **Interpreter** device parameter (described in Table 3.4 on page 66) specifies what rules a printer will use to interpret the stream of input characters in order to make marks on the page. If the value of the **Interpreter** parameter is something other than /PostScript, the printer is being asked to emulate the functions of some other printer.

For example, the Diablo®630 is a daisy wheel printer which has very limited capabilities for putting marks on a page. The input stream is code for characters; the printer assumes one character to follow another until a carriage return or line feed is reached.

Thus, to emulate a Diablo630 printer, the code:

```
(%Serial%) <</Interpreter /Diablo630 /Protocol /Raw>> setdevparams
```

gives Diablo630-like functionality to the serial input channel on a PostScript printer that has a Diablo630 emulator. This functionality is invoked at the next job boundary.

The LaserJet 4 emulator, the LaserJet III emulator, the LaserJet IIP emulator, the color version of the HP7475A plotter emulator, and the Diablo630 emulator have parameters that allow the user to specify default values. The emulator parameters can be set with the **setdevparams** operator and read with the **currentdevparams** operator. The LaserJet 4 emulator has a device parameter set called %PCL% that breaks the tradition of naming these sets using the emulator name.

Tables 3.36 through 3.40 describe the parameters for the LaserJet 4 emulator, the LaserJet III emulator, the LaserJet IIP emulator, the color version of the HP7475A plotter emulator, and the Diablo630 emulator.

Table 3.36 describes the parameters for the LaserJet 4 emulator.

Table 3.36 *Parameters present for the %PCL% device (LaserJet 4 emulator)*

Key	*Type*	*Semantics*
MaxPermanentStorage	integer	This parameter specifies the maximum amount of memory to be dedicated for use by the PCL emulator. **Legal values:** Product-dependent integer. **Errors:** **rangecheck**, **typecheck**
Type	name	*(Read-only)* **Type** has the value of Emulator. For the general definition of **Type**, see Table 3.3 on page 59. **Legal value:** /Emulator **Errors:** None.

Table 3.37 describes the parameters for the LaserJet III emulator.

Table 3.37 *Parameters present for the %LaserJetIII% device (LaserJet III emulator)*

Key	*Type*	*Semantics*
Copies	integer	This parameter specifies the default number of copies of a document to be printed.
Duplex	integer	This parameter sets the initial state of duplexing within a PCL job for printers which are capable of duplex operation. Language commands within the print stream can override the setting of this parameter. Acceptable values for **Duplex** are listed below. 0 Simplex. 1 Long-edge binding duplex. 2 Short-edge binding duplex. The default value for **Duplex** is 0, which means that duplexing is not performed.
FontFixed	boolean	If *true*, a fixed pitch font (for example, `Courier`) is requested. If it is *false*, a proportional spaced font is requested.
FontHeight	integer	This is the height of the font, applicable to scalable proportional fonts. This value is a point-size quantity, multiplied by 100 to avoid floating-point representation. A font that is 8.5 points in height would have the value “850” in this parameter. Note that this value is only used if the font specified by the combination **FontSource** and **FontNumber** is scalable and proportional.

FontItalic boolean If *true*, an italic or oblique font is requested.

FontNumber integer This parameter selects the default font within the current **FontSource**. Applicable values are determined based upon the **FontSource** and the number of fonts that are available from that font source. Use font numbers found printed on the font sample page. If a **FontNumber** is specified that is outside the range, the value 0 is used instead.

FontPitch integer This is the number of characters-per-inch for mono-space scalable fonts. The value is multiplied by 100 to avoid floating-point representations. Thus, to select a 12-pitch font, use the value 1200. This parameter is only used by the PCL5 interpreter if the font specified by the combination **FontSource** and **FontNumber** is scalable and mono-space.

FontSource integer This parameter selects the source of the desired font.

0	Internal font.
1	Downloaded font.
-1	Used when the default font is not to be selected. If the -1 value is used, then the default font is selected via an obsolete method which uses the parameters **FontFixed**, **FontItalic**, **FontWeight** and **FontTypeface**. If it is not -1, these four parameters are not used to select the default font.

FontSymbolSet integer This value is the equivalent of the Symbol Set code. The applicable values are described in Hewlett-Packard manuals. Note that this value is only consulted if the font specified by the combination **FontSource** and **FontNumber** is an unbound font. There are 35 legal values.

4	OD (ISO-60 Norweg)
6	OF (ISO-25 French)
7	OG (German)
9	OI (ISO-15 Italian)
11	OK (ISO-14 JISASCII)
14	ON (ECMA-94 Latin 1)
19	OS (ISO-11 Swedish)
21	OU (ISO-6 ASCII)
36	1D (ISO-61 Norweg)
37	1E (ISO-4 UK)
38	1F (ISO-69 French)
39	1G (ISO21 German)
51	1S (Spanish)
53	1U (Legal)
75	2K (ISO -57 Chinese)
83	2S (ISO-17 Spanish)
85	2U (ISO-2 IRV)
115	3S (ISO-10 Swedish)
147	4S (ISO-16 Portug)
173	5M (PS-Math)

179	5S (ISO-84 Portug)
202	6J (Microsoft® Pub)
205	6M (Corel VENTURA™ Math)
211	6S (ISO-85 Spanish)
234	7J (Desktop)
269	8M (Math-8)
277	8U (Roman-8)
309	9U (Windows™)
330	10J (PS-Text)
341	10U (PC-8 US)
373	11U (PC-8 DN)
405	12U (PC-850)
426	13J (Ventura Intl)
458	14J (Ventura US)
501	15U (PiFont)

FontTypeface integer — This parameter describes the typeface (for example, Times, Helvetica, Palatino). The integer value (which can be up to 16 bits) comes from a table published by Hewlett-Packard.

FontWeight integer — This value, between -7 and +7, describes the "weight" or "boldness" of the font. -7 is very light and +7 is very bold.

Landscape boolean — If *true*, the default orientation of the page is landscape unless otherwise specified in the PCL description of the page.

LineWrap boolean — If *true*, long lines wrap to the next line. If *false*, long lines are truncated.

MaxLJMemory integer — This parameter specifies the maximum amount of memory that the emulator will ask for from the page allocator to store downloaded fonts and macros. This limit is important because the emulator will acquire memory at the expense of the PostScript interpreter's memory needs, such as VM or the font cache. **MaxLJMemory** is rounded to the nearest multiple of a memory block size (8192 bytes).

PaperSize integer — This parameter sets the paper size to be used within the PCL job. This parameter has results similar to the "paper size command" (`[Esc]&l#A`) within the PCL5 language.

The **PaperSize** parameter can specify any of the supported page sizes available to the LaserJet III printer. In addition, there is a special value, –1, which means "unspecified." This allows the printer to draw paper from the default slot. The paper sizes available to the LaserJet III printer and their associated integer values are listed below.

Value	*Paper Size*	*Dimensions (in default units)*
–1	Unspecified	Default slot
1	Executive	522 x 756
2	Letter	612 x 792

3	Legal	612 x 1008
26	A4	595 x 842
80	Monarch Envelope	279 x 540
81	Com-10 Envelope	297 x 684
90	International DL Envelope	312 x 624
91	International C5 Envelope	459 x 649

Note *In the above tables, a "default unit" denotes 1/72 of an inch.*

The default value of **PaperSize** is –1, indicating "unspecified," the default tray.

TopMargin integer Amount of white space at the top of the page, specified in IPU (1/7200 inch). The default is 3600 (1/2 inch).

Type name **Type** has the value of /Emulator. For the general definition of **Type**, see Table 3.3 on page 59.

VMI integer This parameter specifies the space between lines of text in 1/7200 inch units.

WaitTimeout integer This parameter specifies the wait time-out (in seconds) after which a page is ejected. The default is 30.

Table 3.38 describes the parameters for LaserJet IIP emulator.

Table 3.38 *Parameters present for the %LaserJetIIP% device (LaserJet IIP emulator)*

Key	*Type*	*Semantics*
Copies	integer	This parameter specifies the default number of copies of a document to be printed.
FontFixed	boolean	If *true*, a fixed pitch font is requested. If it is *false*, a proportional spaced font is requested.
FontHeight	real	This parameter specifies the desired font height in 1/72 of an inch units.
FontItalic	boolean	If *true*, an italic (or oblique) font is requested.
FontPitch	real	This parameter is used only if **FontFixed** is *true*. **FontPitch** takes a real number specifying the number of characters per inch.
FontSymbolSet	integer	This parameter specifies the mapping from 7 or 8 bit numbers to glyphs that appear on the page. The value of this parameter is the number associated with this field in a downloaded font.

FontTypeface	integer	The value of **FontTypeface** is the number assigned to a particular font (for example, Times, Helvetica, Palatino). The integer value (which can be up to 16 bits) comes from a table published by Hewlett-Packard.
FontWeight	integer	This parameter specifies the “weight” or “boldness” of desired font. The parameter ranges from -7 to +7, where -7 is very light and +7 is very bold.
Landscape	boolean	If *true*, the initial orientation of the page is landscape instead of portrait.
LinesPerInch	real	This parameter specifies the default value for the “vertical motion index.” This determines the interline spacing (and hence the number of lines on the page).
ManualFeed	boolean	See section 4.11.3 of the *PostScript Language Reference Manual*, *Second Edition*.
MaxLJMemory	integer	This parameter allows the user to limit the amount of memory that the LaserJet IIP emulator will use for its needs. This limit is important because the emulator will acquire memory at the expense of the PostScript interpreter’s memory needs, such as VM or the font cache. Within a given emulation job, the LaserJet IIP emulator will use temporary memory in excess of **MaxLJMemory** to hold fonts and macros.
Type	name	**Type** has the value of /Emulator. For the general definition of **Type**, see Table 3.3 on page 59.
WaitTimeout	integer	The value of **WaitTimeout** (in seconds) is used by the LaserJet IIP emulator as the minimum amount of time the emulator will wait for additional incoming characters before declaring the job finished. A value of 0 indicates to the emulator that it should wait forever. The parameter typically has a default value of 30.

Table 3.39 describes the parameters for the color version of the HP7475A plotter emulator.

Table 3.39 *Parameters present for the color version of the %HP7475A% device (HP7475A plotter emulator)*

Key	*Type*	*Semantics*
ColorSetup	string	This parameter allows the user to change the default pen color. The **ColorSetup** parameter is a string which contains a list of numbers. There must be a multiple of five numbers in the string. Each set of five specifies the pen number (integer), width of the pen’s line in millimeters (real), the red color value (real, between 0 and 1.0), the green color value (real, between 0 and 1.0), and the blue color value (real, between 0 and 1.0).

Type	name	**Type** has the value of /Emulator. For the general definition of **Type**, see Table 3.3 on page 59.

Table 3.40 describes the parameters for the Diablo630 emulator.

Table 3.40 *Parameters present for the %Diablo630% device (Diablo630 emulator)*

Key	*Type*	*Semantics*
AutoLF	boolean	If *true*, automatic line feeding is specified.
BoldFontName	name	This parameter specifies the name of the PostScript language font used for boldface printing when **ECS** is *false*.
ECS	boolean	If *true*, the printer emulates the IBM PC Graphics ECS (extended character set) print wheel. If *false*, the printer emulates the 96-character plastic print wheel.
ECSDataWidth	integer	Selects 7- or 8-bit data when **ECS** is *true*.
Pitch	integer	The font width and initial HMI (Horizontal Motion Index) is determined from **Pitch**. **Pitch** can have a value of 10, 12 or 15. Any other value will result in a **rangecheck** error.
RegFontName	name	This parameter specifies the name of the PostScript language font used for regular printing when **ECS** is *false*.
Type	name	**Type** has the value of /Emulator. For the general definition of **Type**, see Table 3.3 on page 59.

3.6 The Fax Environment Interface

This section describes the facilities available for setting, controlling and examining the fax environment shared by all jobs and all users connected to the fax printer. These facilities fall into two categories: fax device parameters and administrative resources.

3.6.1 Fax Device Parameters

Several device settings can be established that control various aspects of the facsimile transmission and receipt. These are stored within the fax printer in non-volatile memory. They persist through multiple fax jobs and are consistent across multiple users connected to a fax printer. These device settings are accessed by means of the %Fax% and %Calendar% device parameter sets which are accessed through the **currentdevparams** and **setdevparams** operators (see 3.5, “Device Parameters).

3.6.2 The %Fax% Device

The %Fax% device is used to control the general, global operation of the fax capability of the fax printer. The semantics of the various parameters in the %Fax% device parameter set are described in Table 3.41 on page 143.

Table 3.41 *Parameters present in the %Fax% parameter set*

Key	*Type*	*Semantics*
ActivityReport	boolean	If *true*, then an activity report is printed automatically whenever the activity buffer is full. If *false*, then no activity report is printed automatically. Printing of the report does not clear the recorded information; the oldest entries are overwritten with new entries. Reports may always be generated by request from the host computer (see also section 5.8.1, “Administrative Resources”). The default is *false*. **Legal values:** *true*, *false* **Errors:** **typecheck**
DefaultCaptionOn	boolean	This parameter determines whether the default page caption routine will place captions on the fax pages. The default value is *true*. **Legal values:** *true*, *false* **Errors:** **typecheck**
DefaultConfirmOn	boolean	This parameter determines whether confirmation reports are produced by the default procedures. The default value is *true*. **Legal values:** *true*, *false* **Errors:** **typecheck**
DefaultCoversOn	boolean	This parameter determines whether the default page cover procedure should actually generate cover pages. The default value is *true*. **Legal values:** *true*, *false* **Errors:** **typecheck**
DefaultResolution	integer	If the value of **FaxType** in the **FaxOptions** dictionary is *null*, then the value of this parameter determines the resolution of the raster fax being prepared. It must have a value of 0 or 1; 1 is the default and specifies to use fine ITU group 3 resolution. **Legal values:** 0, 1 **Errors:** **rangecheck**, **typecheck**

DefaultRetryCount integer This integer parameter is the number of retries attempted if the value of **MaxRetries** in the options dictionary is *null*. It must be in the range of 0 to 100 inclusive. The default value is 0.

Legal values: Any integer in the range of 0 to 100 inclusive.

Errors: **rangecheck**, **typecheck**

DefaultRetryInterval
integer If **RetryInterval** in the options dictionary has a value of *null*, this value is used as the number of minutes to wait between retries. It has a default value of 3, and must be in the range of 1 to 60.

Legal values: Any positive integer in the range of 1 to 60.

Errors: **rangecheck**, **typecheck**

DialingPrefix string The value of this parameter is prepended to all dialing strings supplied by fax jobs before sending those strings to the modem's autodialer. This parameter may find a number of uses, for example to contain a long distance carrier access code or to contain a string ("t") to always select tone dialing. The standard default for this parameter is the empty string.

Legal values: Any string up to 50 characters in length.

Errors: **rangecheck**, **typecheck**

DialToneWaitPeriod
integer At the onset of dialing, and whenever a "w" is encountered in the dialing string, the fax printer waits for a dial tone. This parameter indicates the maximum number of seconds to wait each time. If no dial tone is heard within this period of time, the wait is considered a failure. The default value is 1.

Legal values: Any positive integer in the range of 1 to 10.

Errors: **rangecheck**, **typecheck**

Group3Adjustment integer This parameter determines how received raster faxes are processed for printing.

A value of 0 causes the received raster to be upsampled 2-to-3 (for fine mode). If the resulting image is too long to fit on one page of the selected medium, a continuation page will be printed containing the remainder of the page, assuming it is at least 1/10 inch long. The medium is selected using **PageSize Policy** 5.

A value of 1 invokes a method that treats the raster as if it were 300x300 dpi (for fine raster data). The net effect of this is that pages are printed at approximately two thirds original size with no continuation pages—even for legal on letter. The default medium for the **FaxReceived** device is used.

The default value is 2. A value of 2 causes the raster to be scaled for best fit on a single page that is selected using **PageSize Policy** 5. Halftones may be severely marred by this technique.

Values of 10001 and above are used to invoke a user-programmable procedure. The procedure **ProgrammableGroup3Adjustments** contained in the writeable **ProcSet** instance **FaxDefaultProcs** is called. When this procedure is called, a dictionary on the dictionary stack contains the following entries describing the raster needing to be printed:

firstNonblank Integer line number of first non-blank line in the raster data.

lastNonBlank Integer line number of last non-blank line.

nRows Integer number of scan lines in the raster data.

highres Boolean saying whether the raster data is in fine mode or not.

strm A file object that may be read to obtain the raster data.

Legal values: Any integer in the range of 0 to 20000.

Errors: **rangecheck**, **typecheck**

ID string Provides the string by which the fax machine identifies itself to other fax machines. It is usually set to the company name or the telephone number of the line to which the machine is attached. The string can have up to 20 characters and is defined according to the 1988 CCITT fax protocol.

Legal values: A string of up to 20 characters.

Errors: **limitcheck**, **typecheck**

LocalLanguage string This parameter provides the name of the natural language to use when printing transmission reports and activity logs. The default is (English). The other values supported by all version 2014 (or greater) products with fax support are (Dutch), (French), (German), (Italian) and (Spanish). Translation dictionaries for other languages may also be loaded into the printer. When a

report is to be generated, if a translation dictionary for the named language is not present, the one for English will be used. For more information on translation dictionaries see section 5.8.2.

Legal values: A string of up to 50 characters.

Errors: **limitcheck**, **typecheck**

MaxFaxBuffer integer

This parameter sets an upper bound on the number of bytes of printer RAM that may be used for incoming and outgoing fax data. It is only relevant if **StorageDevice** has a value of (%ram%). This is only an upper limit; no space is set aside based on this parameter's value. The minimum value for **MaxFaxBuffer** is 350000. The default and maximum values are product specific and may be a function of the actual amount of memory in the fax printer.

Legal values: Product-dependent.

Errors: **rangecheck**, **typecheck**

PostScriptPassword string

This password is used by the receiving machine to determine if the sender is authorized to send PostScript language programs. In certain situations, a particular machine may only want to receive PostScript language programs from certain other machines. For this reason, the **PostScriptPassword** exists. It has a maximum length of 32 characters. Its current value is not returned by the **currentdevparams** operator. Instead the string (INVALID) is always shown as the value associated with this key. Attempts to set the **PostScriptPassword** to the string (INVALID) will be ignored.

When a telephone connection is established and the receiving machine is willing to accept PostScript language files (that is, **ReceivePostScript** is *true*), the sending machine is asked to encrypt some arbitrary value (issued by the receiver) using the **PostScriptPassword** from the options dictionary passed to the **faxsendps** operator. The receiver then encrypts the arbitrary value with its own **PostScriptPassword** and compares the encrypted results.

If the encrypted results match, the PostScript language file transmission can take place. If the encrypted results do not match, the receiver refuses to accept the PostScript language file transmission from the sender and the telephone connection is broken. Depending upon the value of **RevertToRaster** in the **FaxOptions** dictionary of the sender, the sending machine may or may not revert to a raster image fax transmission at this point.

This password mechanism can be sidestepped if the **PostScriptPassword** value in the receiving machine is the empty string. In this case, all jobs received in PostScript language form are accepted (if the value of **ReceivePostScript** is *true*). However, when they are run, attempts to change

system or device parameters will fail unless accompanied with a password equal to the **SystemParamsPassword**. If the **SystemParamsPassword** is not set, attempts to change system or device parameters will fail. Those jobs will also not be able to generate outbound faxes themselves or to access any of the **FaxAdminOps** facilities.

Legal values: A string of up to 32 characters.

Errors: **invalidaccess**, **limitcheck**, **typecheck**

ProtocolVersion string (*Read only*) This parameter reports the version number of the T.30 fax code present in the printer.

Legal values: string

Errors: None.

ReceivePostScript boolean If *true*, the machine is willing to receive and execute fax jobs that are PostScript language files. This is the default. If *false*, then any attempt to send PostScript language files to this machine is rejected. Only faxes in ITU compressed raster form are accepted.

Legal values: *true*, *false*

Errors: **typecheck**

Rings integer In order to allow a telephone to be shared by the fax printer and a person, the printer needs to let the phone ring several times to give the person time to answer. If the phone is not shared, it is desirable for the printer to answer the phone as quickly as possible. The integer value supplied with this key determines how many rings of the telephone are to be ignored before the fax printer answers. For example if the value is 2 the printer will attempt to answer the phone just after the second ring.

Legal values: Any integer in the range of 1 to 10.

Errors: **rangecheck**, **typecheck**

ServiceEnable integer This parameter acts as the master on-off control for fax send and receive functions. The parameter takes an integer value with these meanings:

0	Fax completely disabled.
1	Send only enabled.
2	Receive only enabled.
3	Both send and receive enabled.

The default value is 3. If receive is not enabled, then the ringing phone will not be answered. If send is not enabled, then execution of **faxsendps** or any execution of **setpagedevice** that attempts to establish a /Fax **OutputDevice** will fail and a PostScript **ioerror** will be raised.

Legal values: 0, 1, 2, or 3

Errors: **rangecheck**, **typecheck**

Speaker integer This parameter controls the use of the speaker associated with the fax modem. The use of the speaker is determined by an integer code:

0	Off at all times.
1	On until a connection is established, 10-20 seconds.
2	On at all times.

The default value is 1 which allows a human to monitor connections being made for both transmissions and receptions.

Legal values: 0, 1, or 2

Errors: **rangecheck**, **typecheck**

StorageDevice string This parameter designates the name of the storage device used to hold received fax data (either PostScript language files or raster) before it is printed, and to hold outgoing fax data (either PostScript language files or raster) before it is transmitted. The default is a value of (%ram%) which means that printer RAM is to be used. For devices other than %ram% (such as a disk), a parameter set must exist and the parameters **Mounted** and **Writeable** must be defined and set to *true*.

If there is a disk on the fax printer, setting this parameter to its name (for example, (%disk0%)) will mean that the disk is used for storage. Because there is typically more space available on disk than in RAM, this will mean that longer faxes can be received and that longer faxes can be transmitted with just one phone call. Changes to this parameter do not take effect until the fax printer is power-cycled or rebooted.

Legal values: String name of any writeable storage device.

Errors: **ioerror**, **limitcheck**

Type name (Read only) This parameter always has a value of /Parameters. For the general definition of **Type**, see Table 3.3 on page 59.

Legal value: /Parameters

Errors: None.

WaitForDialTone boolean This parameter designates whether the fax printer will wait for a dial tone before it starts to dial. If **WaitForDialTone** is *false*, the call is dialed regardless of whether a dial tone is heard. If **WaitForDialTone** is *true*, then the fax printer will listen (for up to **DialToneWaitPeriod** seconds) for a dial tone. If one is heard, dialing will start. If one is not heard, then the call is considered a failure similar to a busy signal (that is, the retry count is decremented and so forth). The default value is *true*.

Legal values: *true*, *false*

Errors: **typecheck**

3.6.3 The %FaxJobs% Device

Received faxes, transmission reports and certain other fax related print jobs are presented to the I/O serializer on an internal channel known as /FaxJobs. The %FaxJobs% parameter set can be used to control back channel messages from such jobs. Table 3.42 lists the parameters present in the %FaxJobs% parameter set.

Table 3.42 *Parameters present in the %FaxJobs% parameter set*

Key	*Type*	*Semantics*
Enabled	boolean	This parameter must have the value of *true* for back channel messages to be generated on the device specified by the **OutputDevice** parameter. **Enabled** has a default value of *true*. **Legal values**: *true*, *false* **Errors**: **typecheck**
Interpreter	name	(*Read only*) This parameter always has a value of /PostScript. **Legal values**: /PostScript **Errors**: None.
On	boolean	This parameter must have the value of *true* for back channel messages to be generated on the device specified by the **OutputDevice** parameter. The default value of this parameter is *true*. **Legal values**: *true*, *false* **Errors**: **typecheck**

OutputDevice string This parameter is used to give the name of the communication device to be used for back channel messages (for example, %SerialB% or %ScsiComm%). If this string does not contain the name of such a device, then the system default back channel will be used. The default value of this parameter is ().

Legal values: String of up to 32 characters.

Errors: **typecheck**, **rangecheck**

PrinterControl name (*Read only*) This parameter always has a value of /PSPrinter.

Legal values: /PSPrinter

Errors: None.

Type name (*Read only*) This parameter always has a value of /Communications.

Legal values: /Communications

Errors: None.

3.6.4 The %Calendar% Device

The fax printer has a battery powered time-of-day clock that is used to provide the time for cover sheets and scheduling fax transmissions. This clock must be set once initially, and then twice a year to follow daylight savings time.

The string (%Calendar%) identifies the calendar device and the entries in the dictionary describe the local date and time. The entries in the dictionary are described in Table 3.43 on page 150.

Table 3.43 *Parameters present in the %Calendar% parameter set*

Key	*Type*	*Semantics*
Day	integer	This parameter represents the day of the month. **Legal values:** An integer in the range of 1 to 31. **Errors:** **rangecheck**
Hour	integer	This parameter represents the hour. **Legal values:** An integer in the range of 0 to 23. **Errors:** **rangecheck**

Minute integer This parameter represents the minute.

Legal values: An integer in the range of 0 to 59.

Errors: **rangecheck**

Month integer This parameter represents the month.

Legal values: An integer in the range of 1 to 12.

Errors: **rangecheck**

Running boolean This parameter turns the clock off and on. When turning the clock on (setting the value to *true*), the time elements should also be set at the same time in order to avoid a **rangecheck** error.

The clock must be on in order to set the time. If the clock is turned off (to preserve battery power) or is assumed to be inaccurate, the time returned is January 1, 1980 00:00:00.

Legal values: *true*, *false*

Errors: **rangecheck**

Second integer This parameter represents the second.

Legal values: An integer in the range 0 to 59.

Errors: **rangecheck**

Year integer This parameter represents the year. The value of this parameter returned by **currentdevparams**, has special significance. If it is non-zero and in the range 1980 to 2079, then it represents the year. If it is 0, then the clock is either turned off (to preserve battery power) or is assumed to be inaccurate.

Legal values: An integer in the range 1980 to 2079.

Errors: **rangecheck**

CHAPTER 4

Resources

In Level 2, PostScript language objects such as fonts, patterns, filters and so on can be managed as open-ended collections of resources grouped into categories. A resource is requested by giving the resource category and name. If the resource does not reside in VM, the resource management mechanism loads it from an external source, such as a disk, a ROM cartridge, or a network file server. Named resources are discussed in section 3.9, “Named Resources,” in the *PostScript Language Reference Manual, Second Edition.*

The resources listed in Tables 4.1, 4.2, and 4.3 are typically present in all implementations of PostScript language version 2015 and later.

4.1 Regular Resources

Regular resources, listed in Table 4.1, are resources whose instances are ordinary, useful objects, such as font or halftone dictionaries.

Table 4.1 *Regular resources*

Category name	*Instances*
CIDFont	Only some products will have instances defined. These will mainly be products for Japanese and other Asian markets.
CMap	Only some products will have instances defined**.** These will mainly be products for Japanese and other Asian markets.
Font	Product-specific. A product with 35 fonts might list: AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic

	Courier Courier-Bold Courier-BoldOblique Courier-Oblique Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique Helvetica-Narrow-Oblique Helvetica-Oblique NewCenturySchlbk-Bold NewCenturySchlbk-BoldItalic NewCenturySchlbk-Italic NewCenturySchlbk-Roman Palatino-Bold Palatino-BoldItalic Palatino-Italic Palatino-Roman Symbol Times-Bold Times-BoldItalic Times-Italic Times-Roman ZapfChancery-MediumItalic ZapfDingbats	
Encoding	**ISOLatin1Encoding**	**StandardEncoding**
Form	No instances defined.	
Pattern	No instances defined.	
ProcSet	**Test** **StartPage** **FaxOps** **FaxAdminOps** **FaxDefaultProcs** **ColorRendering**	 See section 2.2.3. See section 5.8.1. See sections 2.3.5 and 5.8.2. See section 5.4.
ColorSpace	No instances defined.	
Halftone	**DefaultHalftone**	
ColorRendering	**DefaultColorRendering**	

OutputDevice	**Default**	**Printer**
	Fax	**FaxReceived**
	See section 4.4, “Accessing Product Page Device Capability Information.”	
HWOptions	See section 4.5, “Accessing Product Hardware Options Information.”	
Localization	See section 4.6, “Accessing Information on Natural Languages Supported by the Product.”	
ControlLanguage	See section 4.7, “Accessing Information on Languages Interpreted by the Product.”	
PDL	See section 4.7, “Accessing Information on Languages Interpreted by the Product.”	

4.2 Implicit Resources

Implicit resources, listed in Table 4.2, are resources whose instances are not objects, but which represent some built-in capability of the PostScript interpreter.

Table 4.2 *Resources whose instances are implicit*

Category name	*Instances*	
Filter	**ASCII85Decode**	**ASCII85Encode**
	ASCIIHexDecode	**ASCIIHexEncode**
	CCITTFaxDecode	**CCITTFaxEncode**
	DCTDecode	**DCTEncode**
	LZWDecode	**LZWEncode**
	NullEncode	**RunLengthDecode**
	RunLengthEncode	**SubFileDecode**
ColorSpaceFamily	**CIEBasedA**	**CIEBasedABC**
	CIEBasedDEF	**CIEBasedDEFG**
	DeviceCMYK	**DeviceGray**
	DeviceRGB	**Indexed**
	Pattern	**Separation**
Emulator	**LaserJetIII**	**LaserJetIIP**
	HP7475A	**Diablo630**
	EpsonFX850	**ProprinterXL**
IODevice	**%Serial%**	**%Serial_NV%**
		%Serial_Pending%
	%Parallel%	**%Parallel_NV%**
		%Parallel_Pending%
	%ScsiComm%	**%ScsiComm_NV%**
		%ScsiComm_Pending%
	%LocalTalk%	**%LocalTalk_NV%**

	%LocalTalk_Pending%
%EtherTalk%	**%EtherTalk_NV%**
	%EtherTalk_Pending%
%LPR%	**%LPR_NV%**
	%LPR_Pending%
%AppSocket%	**%AppSocket_NV%**
	%AppSocket_Pending%
%Telnet%	**%Telnet_NV%**
	%Telnet_Pending%
%RemotePrinter%	**%RemotePrinter_NV%**
	%RemotePrinter_Pending%
%PrintServer%	**%PrintServer_NV%**
	%PrintServer_Pending%
%LAT%	
%SNMP%	
%SysLog%	
%TCP%	
%UDP%	
%IP%	
%SPX%	
%IPX%	
%EthernetPhysical%	
%TokenRingPhysical%	
%disk0%…%diskn%	
%cartridge%	
%rom%	
%ram%	
%os%	
%Engine%	
%Console%	
%Scsi%	
%Ide%	
%Fax%	
%FaxJobs%	
%Calendar%	
%PCL%	
%LaserJetIII%	
%LaserJetIIP%	
%Diablo630%	
%HP7475A%	

ColorRenderingType	1
FMapType	2, 3, 4, 5, 6, 7, 8, 9
FontType	0, 1, 3, 4, 5, 6, 9, 10, 11, 32, 42
FormType	1
HalftoneType	1, 2, 3, 4, 5, 6, 9, 10, 100
ImageType	1
PatternType	1

4.3 Resources Used to Define New Resources

Resources used in defining new resources, listed in Table 4.3, can be used to create new resource categories, each containing an independent collection of named instances. This is accomplished through a level of recursion in the resource machinery itself.

Table 4.3 *Resources used in defining new resource categories*

Category name	*Instances*	
Category	**Category**	**ColorRendering**
	ColorRenderingType	**ColorSpace**
	ColorSpaceFamily	**Emulator**
	Encoding	**Filter**
	FMapType	**Font**
	FontType	**Form**
	FormType	**Generic**
	Halftone	**HalftoneType**
	HWOptions	**ImageType**
	IODevice	**OutputDevicePattern**
	PatternType	**ProcSet**
Generic	No instances defined.	

4.4 Accessing Product Page Device Capability Information

The resource category **OutputDevice** has been added to perform the following tasks:

- Enable applications to query product capabilities directly.
- Maintain functional equivalence with Level 1 (where page size capability information was present through enumeration of **letter, legal, a4** or other keys in **userdict).**

The resource category **OutputDevice** is present in interpreters starting with version 2011. This category contains one instance for each **OutputDevice** value that **setpagedevice** can accept for that product. Each instance of this resource category should be a valid key of the page device parameter **OutputDevice**. Products that do not contain the **OutputDevice** page device key — that is, products that have only one possible page device output device — have a single instance for the **OutputDevice** category. This single instance may be **Default** or any product-specific name.

The value of each instance of the **OutputDevice** category is a dictionary that contains key-value pairs describing certain capabilities of that particular output device, such as the possible page sizes or the possible resolutions. This

dictionary does not represent the current state of the PostScript product; it simply provides a static list of some of the possible capabilities of the product. Over time, Adobe is likely to define new entries in this dictionary to reflect added capabilities. In 2015 products and later, the entries listed in Table 4.4 are typically present.

Table 4.4 *Description of keys present in an instance of the category OutputDevice*

Key	*Value*
HWResolution	Array of **HWResolution** values that can be supported by the product. Each element within the array can be either an array of two numbers indicating a discrete **HWResolution** support or an array of four numbers [x1 y1 x2 y2] indicating that the range of hardware resolutions between [x1 y1] and [x2 y2] is supported. Redundant values may be present.
ManualSize	Array of page sizes for the product that can be fed manually. Each element can be either an array of two numbers indicating a discrete page size supported or an array of four numbers [x1 y1 x2 y2] indicating that the range of page sizes between [x1 y1] and [x2 y2] is supported. Redundant values may be present. In a product that does not support the **ManualFeed** page device parameter, the array of page sizes should have no entries.
PageSize	Array of page sizes for the product that can be fed automatically (assuming appropriate media are installed). Each element can be either an array of two numbers indicating a discrete page size supported or an array of four numbers [x1 y1 x2 y2] indicating that the range of page sizes between [x1 y1] and [x2 y2] is supported. Redundant values may be present.
ProcessColorModel	Array of names or strings that indicate the possible colorant models that can be chosen on the product. An element in the array can be one of the following values: /DeviceGray, /DeviceRGB, /DeviceCMYK, /DeviceCMY and /DeviceRGBK.

4.5 Accessing Product Hardware Options Information

The resource category **HWOptions** has been added for the purpose of enumerating those special hardware options that are currently present on a product. The hardware options that are special do not have any other PostScript language facility for indicating that they are present in other than this resource category. For example, a given product might have the ability to support Adobe's PixelBurst™ coprocessor. If the coprocessor is not currently

installed on the product, the **HWOptions** resource would not list **PixelBurst**. When the coprocessor is installed on a given product, **PixelBurst** would appear in the **HWOptions** dictionary. This resource category is optional. Refer to the *PostScript Language Addendum* for the product you are concerned with for complete details of the hardware options available.

Some possible instances of the **HWOptions** resource category are listed in Table 4.5.

Table 4.5 *Possible instances of the HWOptions resource category*

Instance name	*Object type*	*Value(s)*
Fax	string	(USModem) (WorldModem)
Clock	string	(TODClock)
PixelBurst	string	Version information
ColorBurst	string	Version information
Type1Coprocessor	string	Version information

4.6 Accessing Information on Natural Languages Supported by the Product

The resource category **Localization** has been added to allow an application or printer driver some means of determining which natural languages (for example, English, Japanese and German) are supported by a given product. The resources in this category are dictionaries which have (at least) the keys, **Language**, **Country** and **CharSet**; or in short, each dictionary has the localization keys in the %Console% device parameter set (see section 3.5.8, “Console Device Parameters,” on page 133). Each such dictionary represents a legal combination of the values for those keys. It is expected that only a sparse subset of the set of all possible combinations will be supported on any given printer. For example, today one is likely to find the entire category consists of just the combinations:

```
<</Language /EN /Country /US /CharSet /ISO-646-ISV>>
<</Language /JA /Country null /CharSet /JIS-...>>
```

Here, /EN is the code for English, /JA is the code for Japanese, /US is the code for the United States, and the *null* value is used to indicate that no dialect is identified for Japanese. For a complete list of values for **Language**, **Country** and **CharSet**, see Table 3.35 on page page 133.

Each instance in the **Localization** resource category shall have a unique name. The names need not be chosen according to a particular scheme. Although no naming scheme is required, the following discussion is presented to suggest possibilities for naming the instances (dictionaries) that make up the **Localization** resource category.

One scheme for naming the instances is to construct a composite name for the instance based on concatenating the three names in the dictionary separated by hyphens. Thus the two above dictionaries would be named as follows:

```
/EN-US-ISO-646-ISV
/JA--JIS...
```

This approach provides a name that is indicative of the information in the dictionary.

Another alternative, which could be used in place of or in combination with the above scheme is to provide descriptive names for the dictionaries. Thus the above dictionaries might be named as follows:

```
/AmericanEnglish
/Japanese
```

The above naming scheme provides a simple way to change the language. Using either of the above naming schemes (but choosing the second scheme for the example below) the following PostScript code changes the language to Japanese:

```
%set the console to the Japanese localization
(%Console%) /Japanese /Localization findresource setdevparams
```

Note *If the **SystemParamsPassword** is set, you will need to put the **Password** key in the dictionary as well, or run this bit of code as a system administration unencapsulated job. See section 3.1, "Two Kinds of Unencapsulated Jobs," for details.*

4.7 Accessing Information on Languages Interpreted by the Product

The resource category **PDL** has been added to allow an application or printer driver to determine which Page Description Language (PDL) interpreters are available on a given PostScript product. Similarly, the resource category **ControlLanguage** has been added to allow an application or printer driver to determine which control languages are available on a given PostScript product.

These resource categories are present on interpreters starting with version 2015. Each category contains an instance for each language selector available on the PostScript product.

A page description language describes how marks are placed onto media. A control language specifies how the environment and parameter sets and values are configured and determines how jobs are identified. The control language choice also determines the format of printer generated messages on the back channel.

The value of each instance of both the **PDL** category and the **ControlLanguage** category is a dictionary that contains key-value pairs describing one of the languages supported on some channel on the PostScript product. It is possible that Adobe may add new entries to these dictionaries to further identify the language. In 2015 products and later, the entries listed in Table 4.6 are typically present. Table 4.7 gives possible instances of the **PDL** resource category and Table 4.8 gives possible contents of the **ControlLanguage** resource category.

The value of a **Selector** key in a **PDL** category instance corresponds to one of the legal values of the **Interpreter** key in a parameter set of type /Communications. Selectors in the **ControlLanguage** category can be used as the value of the **PrinterControl** key in a parameter set of type /Communications. The value of such keys are set using the **setdevparams** operator.

Table 4.6 *Description of keys present in an instance of the categories PDL and ControlLanguage*

Key	*Type*	*Value*
Selector	name	The name used to select that particular **PDL** or **ControlLanguage** interpreter, such as /PostScript or /LaserJetIII. This same name is used as the name of any parameter set associated with the interpreter. This key is required.
LanguageFamily	string	A string that specifies a family of language levels and versions, such as PostScript or PCL. This key is required.
LanguageLevel	string	A string that specifies what level of the language family is present. This can be a level identifier for languages that have an established leveling, such as 2 (PostScript Level 2) or 5e (PCL 5e), or it can be a product name where no independent level naming system exists. Where no variant of the family exists, this key may be omitted or the associated string may be empty.

LanguageVersion	string	A string that describes the particular version for which the above level of the language family has been implemented. This value is typically a microcode version identifier, such as 2015.101, but it could be a product name, such as LaserJet 4si. If there is no relevant version identifier, this key may be omitted or the associated string may be empty.

Table 4.7 gives possible instances of the **PDL** resource category.

Table 4.7 *Possible instances of the PDL resource category*

Instance name	*Key values*
PostScript	If **Selector** is /PostScript, then **LanguageFamily** is (PostScript), **LanguageLevel** is (2) and **LanguageVersion** is (2015.101).
PCL	If **Selector** is /LaserJetIII, then **LanguageFamily** is (PCL) and **LanguageLevel** is (5).
PCL	If **Selector** is /PCL, then **LanguageFamily** is (PCL), **LanguageLevel** is (5e) and **LanguageVersion** is (LaserJet4).
HPGL	If **Selector** is /HP7475A, then **LanguageFamily** is (HPGL) and **LanguageLevel** is (7475A).
AutoSelect	If **Selector** is /AutoSelect, then **LanguageFamily** is (AutoSelect) and **LanguageVersion** is (2015.101).
HexDump	If **Selector** is /HexDump, then **LanguageFamily** is (HexDump).
ProPrinter	If **Selector** is /ProprinterXL, then **LanguageFamily** is (PPDS) and **LanguageLevel** is (XL).
EpsonGL	If **Selector** is /EpsonFX850, then **LanguageFamily** is (EpsonGL) and **LanguageLevel** is (FX850).
Diablo	If **Selector** is /Diablo630, then **LanguageFamily** is (Diablo) and **LanguageLevel** is (630).

Table 4.8 gives possible contents of the **ControlLanguage** resource category.

Table 4.8 *Possible instances of the ControlLanguage resource category*

Instance name	*Key values*
PostScript	If **Selector** is /PSPrinter, then **LanguageFamily** is (PSPrinter) and **LanguageVersion** is (2015.101).

PJL	If **Selector** is /PJL, then **LanguageFamily** is (PJL) and **LanguageLevel** is (LaserJet4) or (LaserJet4si).

CHAPTER 5

Other Extensions to PostScript Language Level 2

Since the publishing of the *Postscript Language Reference Manual, Second Edition*, there have been and potentially will continue to be extensions to the Level 2 PostScript language. This chapter lists those extensions for versions through 2015.

5.1 Changes to the Halftone Dictionaries

The following section details changes which have been made to the PostScript Level 2 halftone dictionaries.

5.1.1 Changes Affecting All Halftone Types

The following entry was added to the halftone dictionaries of Types 1 through 5.

Table 5.1 *New dictionary entry for all halftone types*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to **findcolorrendering** for it to determine which color rendering dictionary (CRD) to use. The **HalftoneName** key is used by the **GetHalftoneName** procedure that is part of the **ColorRendering** procset.

5.2 New Halftone Dictionaries

The following halftone dictionaries have been added to the Level 2 PostScript language.

5.2.1 Type 6 Halftone Dictionary

This section describes the Type 6 halftone dictionary. For more information about the concepts and terms used below, see “Halftones,” section 6.4 of the *PostScript Language Reference Manual, Second Edition*.

The Type 6 halftone dictionary defines a halftone screen directly by specifying a threshold array at device resolution. This is similar to a Type 3 halftone dictionary, but the threshold array is obtained from a file instead of a string object. This allows threshold arrays to be larger than 65535 bytes (the implementation limit for strings); smaller threshold arrays can also be defined this way.

When presented with a Type 6 halftone dictionary, **sethalftone** reads *width x height* characters from the **Thresholds** file and saves the resulting threshold array in internal storage. The file must supply sufficient data; if it ends prematurely, a **rangecheck** error is raised.

When the current halftone is a Type 6 halftone dictionary, **currenthalftone** returns a halftone dictionary whose **Thresholds** file can be used to access the contents of the current threshold array just as if it were a read-only file. (That is, the **Thresholds** file object returned by **currenthalftone** is different from the one that was given to **sethalftone**.) This file treats the contents of the current threshold array as a circular buffer that can be read repeatedly; it will never reach end-of-file.

Table 5.2 lists the entries in a Type 6 halftone dictionary.

Table 5.2 *Entries in a Type 6 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary. See Table 5.1 for more information.
HalftoneType	integer	(*Required*) Must be 6.
Height	integer	(*Required*) Height of the threshold array, in pixels.
Thresholds	file	*(Required*) When **sethalftone** is used to make a Type 6 halftone dictionary the current dictionary, the next *width x height* characters are read from the file referenced by *file* and become the current threshold array. So *file* must reference a file open for read or read/write access at the time **sethalftone** is called. The *file* object can, of course, be the one returned by the **currentfile** operator. In that case, the next *width x height* characters are read from the input stream and saved as a threshold array. Also, **sethalftone** closes *file* if it encounters an EOF and leaves it open otherwise.
TransferFunction	procedure	(*Optional*) If present, overrides the transfer function specified by **settransfer** or **setcolortransfer**. Required in a Type 6 halftone dictionary that is used as an element of a Type 5 halftone dictionary for a non-primary color component.

Width	integer	(*Required*) Width of the threshold array, in pixels.

5.2.2 Type 9 Halftone Dictionary

This section describes the Type 9 halftone dictionary. For more information about the concepts and terms used below, see "Halftones," section 6.4 of the *PostScript Language Reference Manual, Second Edition*.

The Type 9 halftone dictionary specifies a halftone whose data is proprietary. This type of halftone will be present only in those products whose manufacturers have specifically requested this type of support. If it is not present, attempting to set a Type 9 halftone will result in a PostScript error. It is not possible for PostScript language code to gain any information about contents or appearance of a Type 9 halftone. As a general rule, an application should not explicitly attempt to set a Type 9 halftone. If one is present, it will usually be the default halftone; consequently, any printing that an application does will take advantage of it (unless the application performs its own **sethalftone** call, of course). If it is important to determine whether a Type 9 halftone is being used, check the **HalftoneType** key for the dictionary returned by **currenthalftone**.

Table 5.3 lists the entries in a Type 9 halftone dictionary.

Table 5.3 *Entries in a Type 9 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary. See Table 5.1 for more information.
HalftoneType	integer	(*Required*) Must be 9.

5.2.3 Type 10 Halftone Dictionary

This section describes the Type 10 halftone dictionary. For more information about the concepts and terms used below, see "Halftones," section 6.4 of the *PostScript Language Reference Manual, Second Edition*.

The Type 10 halftone dictionary can be used to specify a threshold array that represents a halftone cell with a non-zero screen angle. Either the Type 3 or Type 6 halftone dictionary can be used to specify a threshold array representing a zero-angle halftone cell, but there is no provision for other angles. Zero-angle halftone cells are easy to specify because they line up nicely with scan lines and because it is not difficult to determine where a

sampled point goes. The Type 10 halftone applies a simple transformation to the halftone cell that converts it into two squares, thus making it easier to specify non-zero angle cells.

Figure 5.1 and Figure 5.2 both illustrate a halftone at 300 dpi with a frequency of 38.4 and an angle of 50.2 degrees. Figure 5.1 shows how this halftone cell can be graphically represented in device space; each asterisk is an (*x*,*y*) coordinate in device space that is mapped to a specific location in the threshold array. Figure 5.2 shows how this cell can be divided into two squares.

Figure 5.1 *Halftone cell graphically represented in device space*

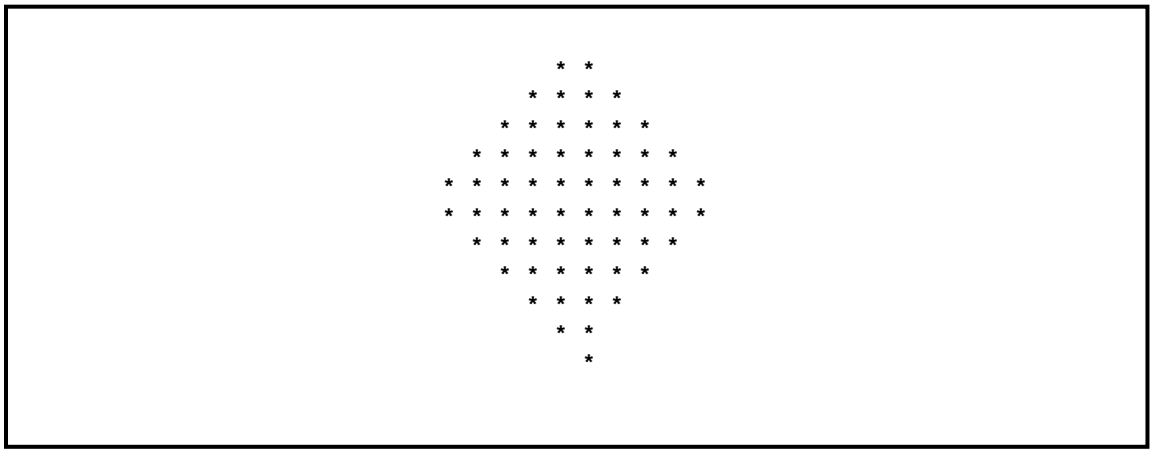
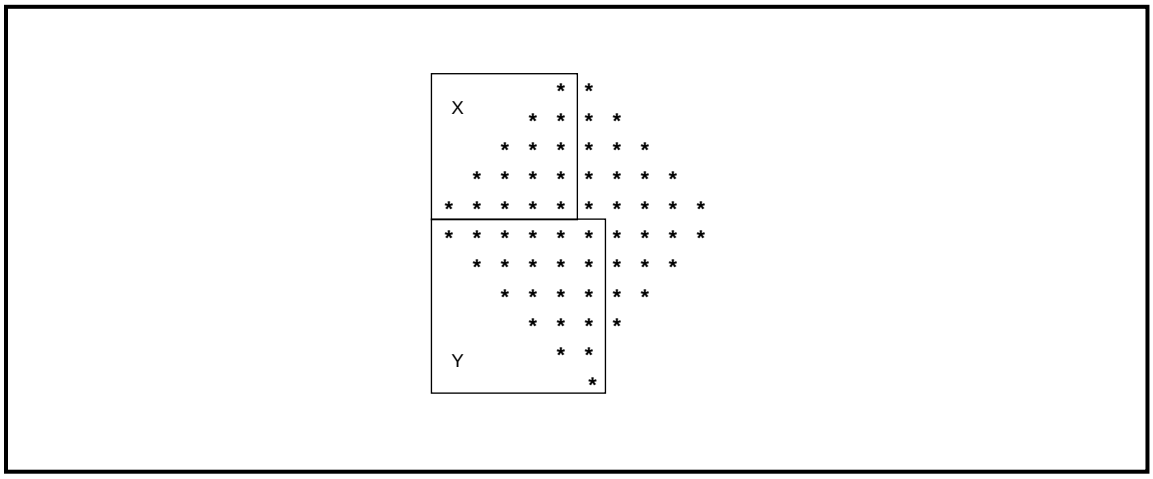
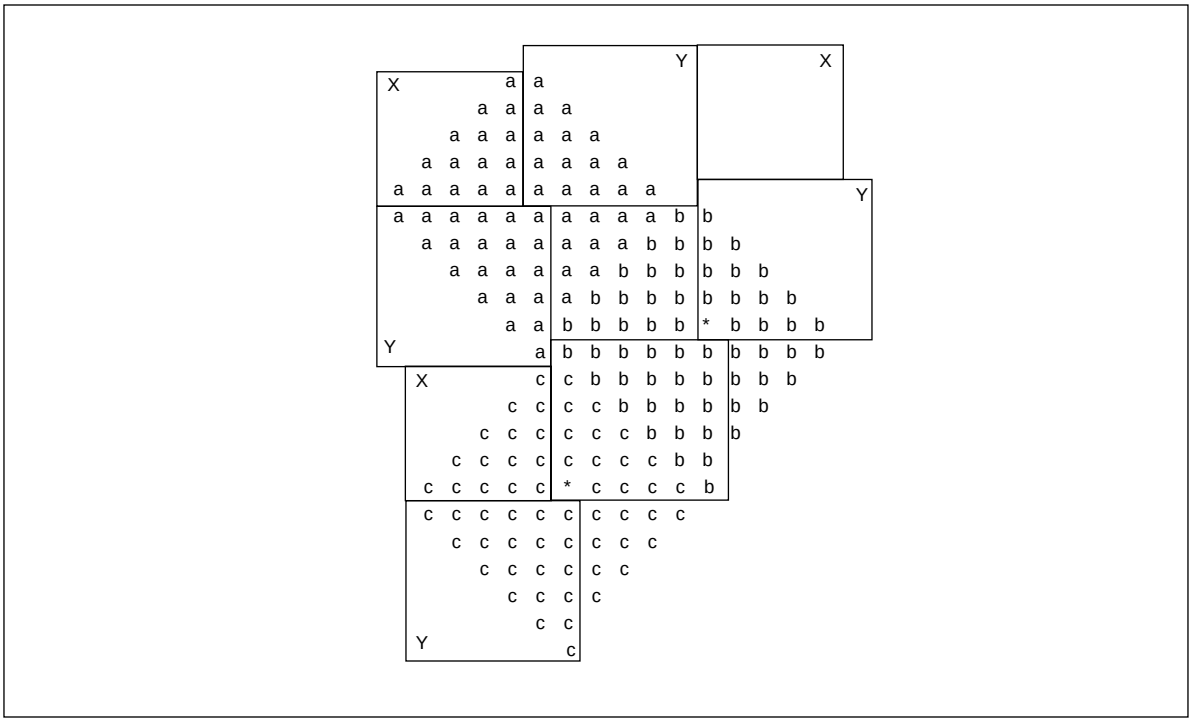


Figure 5.2 *Halftone cell divided into two squares*



If the two squares and the original halftone cell are tiled across device space, the area to the right of the upper square exactly maps into the empty area of the lower square, and the area to the right of the lower square exactly maps into the empty area of the upper square, as Figure 5.3 illustrates.

Figure 5.3 *Halftone cell and two squares tiled across device space*



Any halftone cell will map into two squares in this fashion. The length of one side of the upper X square will equal the distance along the *x* axis from a point in one halftone cell to the corresponding point in the adjacent cell (the asterisks in Figure 5.3 show two such corresponding points). The length of the lower Y square will equal the distance along the *y* axis between the same points.

Looking at this the other way around, a halftone constructed from two squares X and Y (as described above) will have a frequency

$$\text{frequency} = \frac{\text{resolution}}{\sqrt{X^2 + Y^2}}$$

and an angle

$$\text{angle} = \text{atan}\left(\frac{Y}{X}\right)$$

The two squares are much easier to handle and store than the original cell, yet the squares contain the same information. In addition, the squares can be easily mapped into the internal representation used for all rendering.

Table 5.4 lists the entries in a Type 10 halftone dictionary.

Table 5.4 *Entries in a Type 10 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present supplies the name of the halftone dictionary. See Table 5.1 for more information.
HalftoneType	integer	(*Required*) Must be 10.
Xsquare	integer	(*Required*) Length of one side of the upper square.
Ysquare	integer	(*Required*) Length of one side of the lower square.
Thresholds	string or file	(*Required*) Threshold values, specified as either a *string* or *file* object, like Type 3 or Type 6 halftones, respectively, and obeying the same rules. If it is a *string*, it must be **Xsquare** * **Xsquare** + **Ysquare** * **Ysquare** bytes in length. If it is a *file*, the stream must contain at least that many bytes (like Type 6 halftones, it is not necessary for a stream to reach the end-of-file precisely after the requisite number of bytes). In either case, **Thresholds** is ordered with the **Xsquare** square first. The order of pixels is the same as for a sampled image mapped directly onto device space, with the first sample at device coordinates (0,0) and with *x* coordinates changing faster than *y* coordinates. Note that this is the same ordering used by Type 3 and Type 6 threshold arrays. The **Ysquare** data immediately follows the **Xsquare** data and is laid out in the same fashion. **currenthalftone** will return the original dictionary if the value of **Thresholds** is a *string* (like a Type 3 halftone). If the **Thresholds** value in the original dictionary is a *file*, it will be replaced by a new *file* linked to that threshold array (like a Type 6 halftone) in the dictionary returned by **currenthalftone**.

Type 100 Halftone Dictionary

The Type 100 halftone dictionary is almost the same as a Type 9 dictionary, with the exception that the designer of the dictionary may include additional keys. These keys may be used by the printer firmware to specify different halftones under different circumstances. Like a Type 9 halftone, it is impossible for PostScript language code to determine anything about the contents or appearance of the Type 100 halftone. While any optional keys will be visible in the dictionary returned by **currenthalftone**, there is no way to know what they control or what permissible values are. As a result, an application should not explicitly attempt to set a Type 100 halftone. If it is important to determine whether a Type 100 halftone is being used, check the **HalftoneType** key of the dictionary returned by **currenthalftone**.

Table 5.5 *Required entries in a Type 100 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary. See Table 5.1 for more information.
HalftoneType	integer	(*Required*) Must be 100.
other	any type	(*Optional*) Product-specific.

5.3 New Font Types

The following new font types have been added to the Level 2 PostScript language.

5.3.1 Type 32 Font Dictionary

A Type 32 font dictionary is a means for managing device resolution bitmap characters that have been rendered on the host computer prior to transmission to the PostScript interpreter. The host application or driver downloads the bitmaps into the PostScript font cache and manages those characters directly.

This method is a space- and time-efficient alternative to the traditional one for defining fonts as Type 3, where Type 3 **BuildChar** procedure renders the bitmaps using the **imagemask** operator.

For correct results, the host application or driver is required to know certain device-dependent details of the target device, including the resolution and orientation of device space and capacity of the font cache. Therefore, use of Type 32 fonts is appropriate only for attached printers that are under direct control of host software. They are not suitable in a PostScript language document that is intended to be portable.

The primary characteristics of Type 32 fonts include the following:

- Type 32 fonts print with bitmaps of characters and only those glyphs found in the document are rendered and incrementally downloaded to the printer.
- With Type 32 fonts, the host performs rendering based on the resolution of the printer.
- Type 32 fonts occupy storage in the cache only. They do not use VM. Please note that Type 32 fonts differ in this way from Type 3 fonts which utilize storage in the cache and in VM.

- Type 32 fonts are *character identifier* (CID) fonts. For more information on CID fonts, see section 5.6, “CID Font Format,” on page 184.

- Type 32 fonts, though they are host-generated, are compatible with all PostScript printers despite differences in printer resolution and/or printer orientation, as is true with all Adobe fonts and PostScript files. Type 32 fonts can also be rotated and/or scaled.

Note *Rotation and scaling do cause deterioration in the printed image quality of Type 32 bitmap fonts in the same way that they deteriorate the quality of Type 3 fonts.*

Some restrictions apply to Type 32 fonts. In general, embedded encapsulated PostScript (EPS) files and other PostScript code inserted by pass-through in the driver cannot refer to Type 32 fonts defined in the enclosing document. (This restriction is directly due to the limitations of incremental downloading.) Additionally, one cannot perform any transformations that require accessing outline definitions with a Type 32 font. This means that operations like stroking the outline, shadowing and clipping with **charpath** must be performed in the driver or application prior to downloading.

A Type 32 font dictionary must contain the following entries.

Note *Other entries not listed in the table below that are specific to Type 1 fonts (for example, Metrics, Metrics2, CDevProc and PaintType) are not applicable to Type 32 fonts and are ignored.*

Table 5.6 *Entries in a Type 32 font dictionary*

Key	*Type*	*Semantics*
CIDFontType	integer	(*Required*) Must be 4.
FontMatrix	array	(*Required*) Transforms the character coordinate system to the user coordinate system. For more information, see Table 5.1 on page 266 of the *PostScript Language Reference Manual, Second Edition*. In the case of a Type 32 font, the **FontMatrix** should be the inverse of the transformation from default user space to device space of the device for which the bitmap is designed to be used. Initially, the translation components of this matrix should be zero. Positioning of the individual glyph bitmaps is accomplished via metric information provided while defining each glyph. For more information, see the definition of the **addglyph** operator below on page 173.
FontBBox	array	Font bounding box. For more information on **FontBBox**, see Table 5.2 on page 266 of the *PostScript Language Reference Manual, Second Edition.*

FontType integer (*Optional; inserted by* **defineresource**) Must be 32.

New Type 32 Operators

The **addglyph** operator performs incremental downloading for Type 32 fonts. The **addglyph** operator is defined in /BitmapFontInit **ProcSet** rather than in **systemdict**.

addglyph *metric bitmap cid type32font* **addglyph** —

inserts or replaces a Type 32 glyph. The **addglyph** operator installs an image string *bitmap* of a character identified by *cid* to the font cache for the *type32font*. If the old character exists, the old one will be replaced by the new one.

The *metric* operand must be a 6- or 10-number array [w0x w0y llx lly urx ury] or [w0x w0y llx lly urx ury w1x w1y vx vy], respectively, where the elements of the array represent the following:

(*w*0*x*, *w*0*y*)	The width (in real numbers) of the character in writing mode 0.
(*llx*, *lly*)	The lower-left corner of the character bounding box, in character space relative to the character origin; integers only. The difference (*urx* - *llx*) must equal the number of columns in the bitmap. The difference (*ury* - *lly*) must equal the number of rows in the bitmap.
(*urx*, *ury*)	The upper-right corner of the character bounding box; integers only. The difference (*urx* - *llx*) must equal the number of columns in the bitmap. The difference (*ury* - *lly*) must equal the number of rows in the bitmap.
(*w*1*x*, *w*1*y*)	Width of the characters in writing mode 1; real numbers.
(*vx*, *vy*)	Origin 1 relative to origin 0; real numbers. For more information, see Figure 5.6 on page 273 of the *PostScript Language Reference Manual, Second Edition*.

The *bitmap* operand consists of the bitmap data. The bitmap representation is identical to the normal PostScript image representation for a 1-bit per pixel image. Logically, this image is painted in character space with the (0, 0) corner of the image coinciding with (*llx*, *lly*) in character space.

The *cid* operand specifies an integer character identifier (CID).

The *type32font* operand specifies the Type 32 font.

The coordinate system in which metrics and image data are interpreted is character space. When characters are shown at the CTM for which the font was designed, the complete transformation from character space to device space is the identity one-to-one. Thus, the image is treated as a device-resolution bitmap, positioned with the image space origin at (*llx*, *lly*) relative to **currentpoint**.

Errors: **stackunderflow**, **typecheck**, **rangecheck**, **limitcheck**, **invalidfont**

A **rangecheck** error occurs if the *urx* is less than *llx*, *ury* is less than *lly*, or the image dimensions implied by these values are inconsistent with the length of the bitmap string.

A **limitcheck** error occurs if the glyph cannot be placed in the font cache, either because it is too large or because the cache is full.

An **invalidfont** error occurs if the specified font is not Type 32.

removeall *type32font* **removeall** —

removes glyphs defined for the font *type32font*.

The deleted glyphs are removed from the font cache immediately. They may continue to occupy memory until all pages on which those glyphs were used have been printed.

Errors: **invalidfont**, **rangecheck, typecheck**, **stackunderflow**

An **invalidfont** error occurs if the specified font is not Type 32.

removeglyphs *firstcid lastcid type32font* **removeglyphs** —

removes all glyphs identified by CID between *firstcid* and *lastcid*, inclusive in the font *type32font*.

The deleted glyphs are removed from the font cache immediately. They may continue to occupy memory until all pages on which those glyphs were used have been printed.

Errors: **invalidfont**, **rangecheck, typecheck**, **stackunderflow**

An **invalidfont** error occurs if the specified font is not Type 32.

A **rangecheck** error occurs if *lastcid* is less than *firstcid* or if these numbers are outside the valid range of CIDs (0 to 65535). However, no error arises from references to nonexistent glyphs.

Behavior of Painting Operators with Type 32 Fonts

At the time **show** is executed, the font machinery checks to see if the concatenation of the **FontMatrix** (including prior **makefont** transformation if any) and of the CTM is the identity, disregarding translation.

If the catenation of **FontMatrix** and the CTM is identity, the bitmap that was defined by **addglyph** is painted directly on the current page. If not, the bitmap is used as an image source and is painted on the current page with the suitable transformation, as if by the **imagemask** operator. However, if so, the font machinery consults the CMap again for a notdef CID; if the CMap is not present, the font machinery resorts to CID 0. All CID fonts must define a glyph for CID 0 or else an **invalidfont** error will occur. These rules regarding **show** and non-existent glyphs apply to all types of CID fonts.

The **charpath** operator produces an empty path.

5.3.2 Type 42 Font Dictionary

PostScript interpreters can now optionally include support for font Type 42. The Type 42 font format is a TrueType™ font with a PostScript language wrapper to make it conform to the PostScript language font model. A printer driver can distinguish a product that supports font Type 42 by using the PostScript Printer Description (PPD) file to extract the appropriate query. The query looks at the resource instance **FontType** to see if Type 42 is supported. Refer to the **FontType** instance in Table 4.2 on page 155. For more information about the Adobe Type 42 font format, refer to technical note #5012 titled *The Type 42 Font Format Specification*, available from Adobe Developer Support.

The TrueType font format was originally developed by Apple Computer, and is currently supported by the Macintosh® and Windows 3.1 operating environments. Prior to the PostScript language version 2013, documents containing TrueType fonts could only be sent to PostScript interpreters in one of two ways. On products that supported 680X0-class controllers, the TrueType rasterizer could be downloaded prior to processing the TrueType document. Otherwise the TrueType font had to be converted into a PostScript language Type 1 or Type 3 font. Both of these methods have disadvantages: the TrueType rasterizer is large and does not work on all platforms, and conversion to Type 1 or Type 3 is not exact.

5.4 CRD Selection Based On Rendering Intent

There is a new method by which color rendering dictionaries (CRDs) can be selected in a device-independent manner in a page description. This selection accounts for the *rendering intent* of the reproduction, the *device setup*, and the current *halftone*.

A rendering intent is information about the rendering of colors in addition to their colorimetric specification. For example, one might want to specify that a scanned image be rendered in a "*pleasing*" manner, much like a photograph, instead of requiring a colorimetric reproduction. Device setup refers to the state of the device. This information is kept in the **pagedevice** dictionary and consists of parameters such as **MediaType** and **HWResolution**. Halftone information is resident in the graphics state.

A new operator has been created for the purpose of selecting CRDs by rendering intent. The new operator is named **findcolorrendering**.

5.4.1 findcolorrendering

A CRD is found using the **findcolorrendering** operator. This operator is not a standard part of PostScript Level 2. Applications, printer drivers and utilities should test to see if it is *known* on any product prior to trying to use it. It is not meant for use with PostScript Level 1 interpreters.

The syntax of the **findcolorrendering** operator is:

findcolorrendering *renderingintent* **findcolorrendering** *crdname bool*

renderingintent is a name or string specifying the rendering intent. *crdname* is a name representing a CRD present in the **ColorRendering** resource category. If *bool* is true, *crdname* specifies a CRD present in the **ColorRendering** resource category that matches the desired rendering intent, device setup, and halftone combination. If *bool* is false, a CRD satisfying this combination exactly is not available. In this case, *crdname* specifies a substitution for the desired CRD. In either case, the CRD specified by *crdname* can be instantiated in the graphics state by using **findresource** and **setcolorrendering**.

findcolorrendering should be called after all commands that influence either the halftone or the device setup in order to insure that all parameters that may be considered for selection of a CRD are accounted for correctly.

An example usage of **findcolorrendering** that selects a perceptual CRD is shown here:

```
/findcolorrendering where
{
  % findcolorrendering available
  pop
  /Perceptual findcolorrendering
  {
    % CRD found which satisfies combination of
    % rendering intent, device setup, and halftone
    /ColorRendering findresource setcolorrendering
  } {
    % exact match for CRD not found
    % use it, or find a CRD another way
    % in this example we'll use it if it's
    % not DefaultColorRendering
    dup
    /DefaultColorRendering eq {
      pop
    } {
      /ColorRendering findresource setcolorrendering
    } ifelse
  } ifelse
} {
  % findcolorrendering not available
  % in this example we'll use the current CRD
  % so we do nothing
} ifelse
```

This example first checks for the existence of **findcolorrendering**. If found, it uses **findcolorrendering** to attempt to find a CRD for a perceptual rendering intent. If successful, it installs the CRD in the graphics state. If **findcolorrendering** returns false, there are three possible actions:

- Use the substitution CRD that is returned.
- Pick a different CRD using your own method.
- Leave in the graphics state the currently installed CRD.

In this example, a test first is made to see if **DefaultColorRendering** is returned by **findcolorrendering**. In general, this signifies that a useful substitution was not possible. In this case, the best choice is to leave the graphics state's current CRD installed. Installation of the returned CRD is appropriate if the substitution name is different than **DefaultColorRendering**.

The current list of rendering intents recognized by Adobe Systems, Inc. is kept with Adobe's Developer Support organization. Note that not all possible rendering intents will be supported by a particular device. In addition, other devices may support additional rendering intents beyond the standard set. This is a product-dependent decision.

There is purposefully a close correspondence between the Adobe rendering intents and the rendering intents of the International Color Committee (ICC) format. This correspondence is a function of the ICC format version number. Contact Adobe's Developer Support organization for more details.

The initial list of recognized rendering intents and their descriptions are shown in Table 5.7.

Table 5.7 *Rendering intent list*

Rendering Intent	*Description*
AbsoluteColorimetric	Absolute colorimetry is used. For reflection print this means that *y* of unprinted paper (paper white) is less than 1. A colorimetric reproduction is provided for in-gamut colors. Out-of-gamut colors are mapped to the border of the reproducible gamut. This has the advantage of providing exact color matches from printer to printer. It has the disadvantage of causing colors with *y* values between the paper's white and 1 to be out-of-gamut. Example usage would be for spot colors where an exact color reproduction is desired.
RelativeColorimetric	Relative colorimetry is used. For reflection print this means that *y* of paper white is taken to be 1. All colorimetric measurements are normalized based on the paper's colorimetry. A colorimetric reproduction is provided for in-gamut colors. Out-of-gamut colors are mapped to the border of the reproducible gamut. This has the advantage of providing a larger effective gamut so that bright colors will more likely be in-gamut. It has the disadvantage of sacrificing exact color matches for printers with different paper white points. Example usage would be for spot colors where a color reproduction relative to the paper's white is desired.
Saturation	Saturation-relative colorimetry is used. A reproduction in which saturation is emphasized. In-gamut colors may or may not be colorimetric. Example usage would be for business graphics where saturation is the most important attribute of color.
Perceptual	Relative colorimetry is used. A reproduction which provides a perceptual or pleasing appearance. This generally means both in- and out-of-gamut colors are modified from their colorimetric representation. Example usage would be for scanned images.

5.4.2 Relationship to Graphics State Parameters

Currently the only graphics state parameter that is considered by **findcolorrendering** is the halftone. Other parameters of the graphics state like black generation, undercolor removal, and transfer functions are not accounted for since they do not require per object modification. Halftoning requires such modification.

5.4.3 Inside findcolorrendering

findcolorrendering forms the name of a color rendering dictionary from the rendering intent, the device setup, and the halftone. The resulting name takes the form

renderingintent.devicesetup.halftone

where *renderingintent* is taken verbatim from the *renderingintent* operand, and *devicesetup* and *halftone* are found indirectly through procedures resident in the **ColorRendering** instance of the **ProcSet** resource category. *devicesetup* is returned by a call to the **GetPageDeviceName** procedure in the **ColorRendering ProcSet**. *halftone* is returned by a call to the **GetHalftoneName** procedure in the **ColorRendering** procset. The syntax of **GetPageDeviceName** and **GetHalftoneName** are as follows.

GetPageDeviceName - **GetPageDeviceName** *devicesetup* and

GetHalftoneName - **GetHalftoneName** *halftone*

GetPageDeviceName and **GetHalftoneName** always return a name. If they are unable to return a meaningful name, they return /none. Both **GetPageDeviceName** and **GetHalftoneName** may perform a variety of operations in an effort to return a meaningful name. **GetPageDeviceName** uses as an operand to its name selection process the pagedevice key **PageDeviceName** (see page 13). Like **findcolorrendering** itself, **PageDeviceName** is not available in all PostScript Level 2 implementations. In an analogous manner, **GetHalftoneName** uses the optional **HalftoneName** key in the current halftone dictionary. The name selection processes for these two procedures may be as simple as looking for the appropriate name in the appropriate location and returning /none if it is not found. Or it may be considerably more complex. For example, one could analyze the current halftone in terms of angle and frequency to classify it.

If the name formed by the concatenation of rendering intent, device setup, and halftone is not the name of a CRD in the **ColorRendering** resource category, **findcolorrendering** calls **GetSubstituteCRD**. **GetSubstituteCRD** is also located in the **ColorRendering** instance of the **ProcSet** resource category. Its syntax is as follows.

GetSubstituteCRD - *renderingintent* **GetSubstituteCRD** *crdname*

where *renderingintent* is the rendering intent passed to **findcolorrendering**, and *crdname* is the name of a substitution CRD that exists in the **ColorRendering** resource category. When **GetSubstituteCRD** is called, **findcolorrendering** always returns *false* since the desired CRD is not available. **findcolorrendering** returns the CRD returned by **GetSubstituteCRD**. If **findcolorrendering** does not call **GetSubstituteCRD**, it returns *true*. **GetSubstituteCRD** returns **DefaultColorRendering** in the event it cannot generate a meaningful CRD substitution. All PostScript Level 2 interpreters have a CRD named **DefaultColorRendering**.

5.4.4 Modifying the CRD Selection Process

findcolorrendering will reside in **systemdict** for PostScript Level 2 products with ROM versions 2015.100 and beyond. For earlier versions of PostScript Level 2 products, **findcolorrendering** may be downloaded by a utility outside the sever loop, or it may be downloaded on a per job basis. An implementation of **findcolorrendering** and a generic implementation of its associated machinery can be obtained from Adobe's Developer Support organization. However it ends up in the printer, **findcolorrendering** is not meant to be overridden to achieve a modification to the CRD selection process. Its purpose is to delegate portions of this task to **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD**.

It is anticipated that **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD** may be overridden. In addition, it is anticipated that each procedure's specific implementation will vary from device to device to account for the different resident CRDs.

Adobe supplies a baseline version for **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD**. Customizing this baseline version can be done on a product by product basis. This baseline version satisfies the stated requirements for these three procedures as well as providing a template for modifying this selection process.

To aid **GetPageDeviceName** in returning meaningful device setup information, Adobe has added to the page device dictionary a **PageDeviceName** key. In general, **GetPageDeviceName** first looks in the page device dictionary for a **PageDeviceName** key. This key's value can be set using **setpagedevice**. If this key is not present or if its value is *null*, **GetPageDeviceName** constructs a name for the device setup from the current page device parameters, e.g. **MediaType**, or may simply return /none. This fall-back device setup name construction is device-dependent and undocumented. One can override this fall-back construction by replacing **GetPageDeviceName**.

To aid **GetHalftoneName** in returning meaningful halftone information, Adobe is advocating that generators of halftone dictionaries include a **HalftoneName** key. In general, **GetHalftoneName** first looks in the current halftone dictionary for a **HalftoneName** key. If found, this key's value is returned. If not found, **GetHalftoneName** may analyze the current halftone and attempt to form a name or may simply return /none. This fall-back halftone name construction is device-dependent and undocumented. One can override this fall-back construction by replacing **GetHalftoneName**.

5.5 Synchronizing CRDs and ICC Profiles

International Color Consortium (ICC) profiles on the host and PostScript color rendering dictionaries in the printer can contain identical information for color transformations. To reduce printer memory requirements and PostScript file transmission times for color transformations, ICC profiles on the host and CRDs in the printer should be synchronized.

CRDs should be created with a **CreationDate** entry indicating the date and time of CRD creation or most recent modification. The **CreationDate** entry is optional. This date and time information should correspond to the date and time entry of any *companion* profiles. A companion profile embodies the same transformation, but in a different format — for example, profile versus CRD. Date and time information is available from the profile's header and the **calibrationDateTimeTag**. Even if no companion profile is constructed, date and time information should still be supplied in the CRD.

The optional CRD entry **CreationDate** is a PostScript string whose format closely follows that defined by the international standard Abstract Syntax Notation One (ASN.1), defined in CCITT X.208 or ISO/IEC 8824. This string is of the form:

```
(YYYYMMDDHHmmSSOHH'mm')
```

where:

YYYY	Year
MM	Month (01-12)
DD	Day (01-31)
HH	Hour (00-23)
mm	Minutes (00-59)
SS	Seconds (00-59)

O	Relationship of local time to Greenwich Mean Time (GMT) (A plus sign (*+*) indicates that local time is later than GMT, a minus sign (*-*) indicates that local time is earlier than GMT, and *Z* indicates that local time is GMT.)
HH'	Absolute value of the offset from GMT in hours
mm'	Absolute value of the offset from GMT in minutes

Fields after the year are optional. The default values for day and month are 1; all other numerical fields default to 0. If no GMT information is specified, the relationship of the specified time to GMT is considered to be unknown. Whether or not the time zone is known, the date should be specified based on local time.

Profiles should be extended with the optional ICC tag **crdInfoTag**. The **crdInfoTag** tag contains the PostScript-product name to which this profile corresponds and the names of the companion CRDs. (Note that a single profile can generate multiple CRDs.)

The format of **crdInfoTag** is given in Table 5.8.

Table 5.8 *Format of crdInfoTag*

Byte(s)	*Content*
0-3	‘crdi’ (0x63726469) type descriptor
4-7	Reserved, must be set to 0
8-11	PostScript-product name character count, including terminating null
12-*m*-1	PostScript-product name string in 7-bit ASCII
m-*m*+3	Rendering intent 0, CRD name character count, including terminating null
m+4-*n*-1	Rendering intent 0, CRD name string in 7-bit ASCII
n-*n*+3	Rendering intent 1, CRD name character count, including terminating null
n+4-*p*-1	Rendering intent 1, CRD name string in 7-bit ASCII
p-*p*+3	Rendering intent 2, CRD name character count, including terminating null
p+4-*q*-1	Rendering intent 2, CRD name string in 7-bit ASCII

q-*q*+3	Rendering intent 3, CRD name character count, including terminating null
q+4-*r*	Rendering intent 3, CRD name string in 7-bit ASCII

If no companion CRD is available for a given profile, then the character count entry is zero and there is no string.

CreationDate and **crdInfoTag** can be synchronized differently depending on whether bidirectional communications are available between the host and the printer and whether the CRD was supplied with the printer or was downloaded from a host in the field.

Bidirectional communication allows the printer to be queried to determine the availability of a given CRD and its associated **CreationDate**. In the absence of bidirectional communications, the list of printer-resident CRDs and their **CreationDate** entries is available through the printer's PPD and the host profile registry.

PPDs currently contain the names of the CRDs that ship with a printer. In the future, the PPD format will be extended to contain the **CreationDate** entry for each CRD. The registry should be updated whenever CRDs are downloaded. The existence and form of the registry may vary between platforms.

There are three cases to consider:

- *CRDs and ICC profiles are made together.* The driver (or application) determines whether it needs to construct and download a CRD for a given profile in the following way.

  The driver optionally checks whether the profile corresponds to the printer by comparing the PostScript-product name field in the **crdInfoTag** with the printer's product name. The product name for the printer is obtained from the **product** operator or from the PPD. This comparison limits the selection of profiles to only those appropriate for the given printer.

  Based on the desired rendering intent, the driver checks whether the printer has a CRD with the name specified in the **crdInfoTag**. CRDs are located in the **ColorRendering** resource category. If there is a profile that corresponds to the printer product name, the driver compares the profile's date and time field to the CRD's **CreationDate** key. If the two match, it is not necessary to download the profile because the companion CRD already exists. If no CRD with the name specified in the **crdInfoTag** is found, then CRDs are generated from ICC profiles, as described below.

- *CRDs are generated from ICC profiles and then downloaded.* A driver can download CRDs for a particular job, in which case there will be no companion CRDs for this synchronization for subsequent jobs. Alternatively, the driver can make the CRD persistent in the printer by generating a **CreationDate** entry, updating the registry, and updating the profile to have the correct **crdInfoTag**.

- *No profile exists for CRDs in the printer.* This situation occurs primarily with existing CRDs that have no companion profiles. Synchronize a companion profile with CRDs as follows:

  Use the CRD **CreationDate** field, if available, for the date and time field of the profile.

  Alternatively, update the CRD in the printer and registry using the **CreationDate** key corresponding to the date and time field of the new profile.

  In either case, **crdInfoTag** must be filled in correctly. Note also that in this case the CRD updates may be volatile to power cycles of the printer. After such power cycles, the registry should be updated.

5.6 CID Font Format

The PostScript language has been extended to support CID-keyed composite fonts, **CIDFont** resources and **CMap** resources.

An instance of a **CIDFont** resource is a dictionary, often called a *CID font*. CID font dictionaries are a new base font Type. These dictionaries allow large collections of character outlines to be stored in one dictionary. The outlines are accessed by an integer *Character Identifier* (CID) rather than by a glyph name or character code.

An instance of a **CMap** resource is a dictionary often called a *CMap*. Encoding information is stored in CMap dictionaries. CMap dictionaries define the mapping from character codes to a font number and a character selector. A character selector is a CID, a character code or a glyph name. Single byte, multi-byte and mixed single and multi-byte encodings are supported.

An additional **FMapType** is now supported with composite fonts. This **FMapType** allows CMap dictionaries to be used to define the encoding and allows CID fonts to be used as base fonts.

For more information, refer to the document *Adobe CMap and CIDFont Files Specification,* dated 11 June 1993, Technical Note 5014, and *Adobe CID-Keyed Font PostScript Language Extensions*, which are both available through Developer Support.

5.6.1 CIDFont and CMap Resource Categories

A **CIDFont** resource instance is a dictionary especially well suited for representing a large set of glyph outlines. The glyph outlines may be defined as Type 1 **CharStrings**, PostScript **BuildGlyph** procedures, or TrueType glyph procedures. An integer character identifier (CID) is used to access glyph outlines in CID fonts. The mapping from CID to a glyph is defined by the registry and ordering information. Different language groups may have the glyphs in a different order. Several registries and glyph orders have been defined for public use.

A **CMap** resource instance is a dictionary that defines mappings from character codes (single or multiple byte) to CIDs or other character selectors and a font number. The CMap is used in conjunction with one or more CID fonts or base fonts. The CIDs it produces can select glyphs from a **CIDFont**; the other selectors (codes or names) can select glyphs from an ordinary base font. The font number selects a font from the **FDepVector** array in a composite font.

A composite font (**FontType** 0) with **FMapType** 9 combines a **CMap** dictionary with one or more CID font or base font dictionaries. The **CMap** entry in the composite font dictionary specifies the CMap, and the **FDepVector** array specifies the CID fonts or base font dictionaries to be used.

The resource operators **resourceforall**, **resourcestatus**, **findresource**, **defineresource**, and **undefineresource** can be used to list, acquire status, find, define, and undefine **CIDFont** or **CMap** resource instances. If the instance is automatically loaded from an external source, it is loaded into global VM. Otherwise it is built in the VM allocation mode active when the **CIDFont** or **CMap** instance is created.

CID font dictionaries can be used with **makefont**, **scalefont**, **selectfont**, **setfont** and **currentfont** operators. However when a CID font is the current font only **glyphshow** is allowed. **glyphshow** will now take as an argument an integer as well as a name object. When the current font is a CID font the integer will be used as the CID to find and show the glyph.

The show operators **show**, **ashow**, **widthshow**, **xshow**, **xyshow**, **yshow**, **stringwidth**, **cshow** and **kshow** are not supported when the current font is a CID font. Also **findfont** or

```
/Font findresource
```

cannot be used to load CID fonts into VM, nor will

```
/Font resourceforall
```

list CID fonts. The **CIDFont** resource category is independent of the **Font** resource category.

5.6.2 Extensions to Existing Operators

glyphshow *name* **glyphshow** -
int **glyphshow** -

If the current font is a CID font, then the argument must be an integer object. The integer is used as the CID to find and show the character in the CID font. A **typecheck** error is raised if the element on the stack is an integer and the current dictionary is not a CID font or the current dictionary is a CID font and the object on the stack is not an integer. An **invalidfont** error is raised if **glyphshow** is executed when the current dictionary is a composite font (Type 0).

Errors: **invalidaccess**, **invalidfont**, **nocurrentpoint**, **stackunderflow**, **typecheck**

show *string* **show**
ashow *ax ay string* **ashow -**
widthshow *cx cy char string* **widthshow -**
stringwidth *string*
stringwidth *wx wy*
etc. {*other operators in show family except* **cshow** *and* **glyphshow**}
If the current font is a composite font with **FMapType 9**, the **CMap** mapping algorithm described in the extensions to composite fonts is applied to select the glyph.

Errors: Refer to the *Postscript Language Reference Manual*, *Second Edition.*

cshow *proc string* **cshow** -

To maintain compatibility with existing PostScript files, **cshow** has been modified for composite fonts that contain CID fonts as base fonts. When the base font is a CID font, the code put on the stack for execution of *proc* is the low order 8 bits of the character code from the *string.* The original code is stored in an internal variable. If the *proc* does not change the current font but executes a show operator, the glyph is selected by using the original character code and the root composite font as the current font. For example, if the input string to **cshow** is <2240>, the code on the stack when *proc* is executed would be 40 (hexadecimal). If *proc* put this value into a 1 byte string and did a show, the string <2240> would be used to look up and show the glyph from the root composite font. A **rangecheck** error would occur if *proc* tried to

show a string with a byte other than 40 (hexadecimal). A **rangecheck** error is raised if a show operator executed by *proc* uses a value other than the code on the stack when *proc* is invoked.

Errors: **invalidaccess**, **invalidfont**, **nocurrentpoint**, **rangecheck**, **stackunderflow**, **typecheck**

5.6.3 New Operator

composefont *name cmap array* **composefont** *font*

This operator generates a composite font made from the *cmap* and the fonts or CID fonts listed in the *array*. The *array* can contain names or the actual dictionaries. The *cmap* can either be a name or a **CMap** resource instance. The composite font that is created will have an **FMapType** entry set to 9 and a **CMap** entry set to the **CMap** resource instance. The **FDepVector** will be set to the array of fonts or CID fonts. The **Encoding** entry will be set to the identity mapping and will have the same number of elements as the **FDepVector**. The **FontName** of the composite font will be set to the name on the stack when **composefont** was executed. A /Font defineresource will be executed by **composefont**. This will associate the **FontName** with the font dictionary in the **FontDirectory** dictionary. This will overwrite any previous definition of a font with the same name. The **composefont** operator will always create a new font dictionary regardless whether one exists that is made from the same CMap and array of fonts. It is recommended that **composefont** be used to create a composite font and then use **findfont** to retrieve it from the **FontDirectory**. However, the contents of **FontDirectory** are subject to **save** and **restore**. Therefore **composefont** must be used within the **save restore** before the first use of **findfont**.

Errors: **limitcheck**, **rangecheck**, **dictfull**, **invalidfont**, **stackunderflow**, **typecheck**, **invalidaccess**

5.7 Additional CIE-Based Color Spaces

Whereas PostScript language versions 2015 and earlier supported only device-dependent CMYK colors, the PostScript language now supports calibrated CMYK colors as well. This new feature is an extension to the CIE-based color spaces described in section 4.8.3 of the *PostScript Language Reference Manual, Second Edition*.

Two New Color Spaces: CIEBasedDEF and CIEBasedDEFG

Two new CIE-based names have been added for use with the **setcolorspace** operator. They are **CIEBasedDEFG** and **CIEBasedDEF**.

CIEBasedDEF and **CIEBasedDEFG** are 3- and 4-component color spaces which extend the Adobe CIE-based color spaces to include other additional color spaces, in particular calibrated CMYK color spaces. The additional color spaces supported as of PostScript language version 2016 include the following:

- CIE 1976 L*u*v and calibrated RGB from scanners, both of which are 3-component color spaces; and
- Calibrated CMYK, which is a 4-component color space.

CIEBasedDEFG and **CIEBasedDEF** are simple pre-extensions to the **CIEBasedABC** color space. The following figure (Figure 5.4) shows how the **CIEBasedABC** color space looked prior to the addition of the **CIEBasedDEF** and **CIEBasedDEFG** pre-extensions.

Figure 5.4 *The original CIEBasedABC structure*

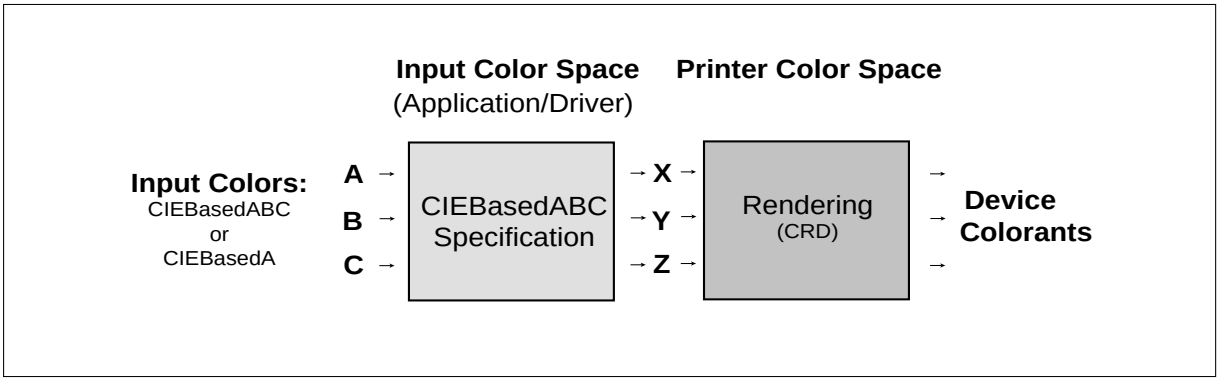


Figure 5.5 shows how the new pre-extension fits with the pre-existing **CIEBasedABC** color spaces.

Figure 5.5 *The new CIEBasedABC pre-extension*

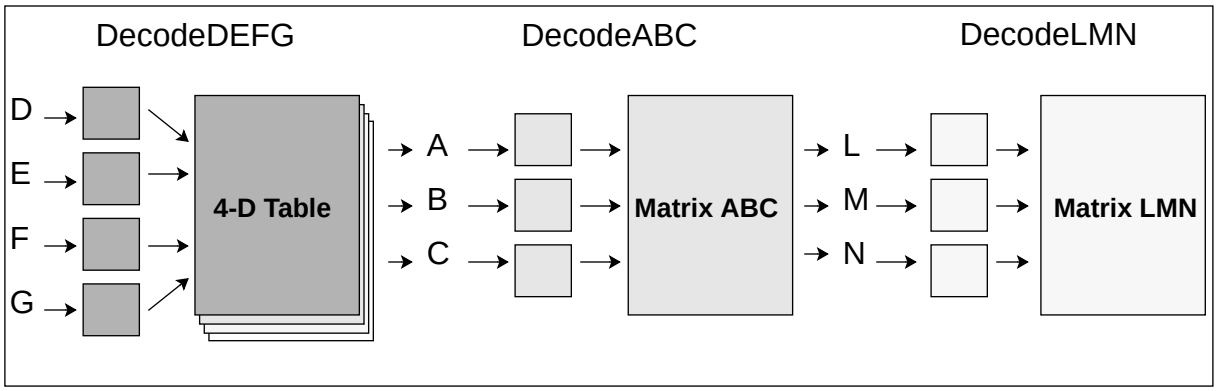


Figure 5.5 represents a 4-component color space transformation. A 3-component color space transformation would be structured identically except that a 3-input/3-output table would function where the 4-input/3-output (4-D) table appears in the diagram above and the procedure **DecodeDEF** would be applied instead of **DecodeDEFG**.

As Figure 5.5 shows, the components of the input colors (*D*, *E*, *F*, *G*) are first transformed component-wise in the color space pre-extension (**CIEBasedDEFG** specification). This transformation is performed by applying the PostScript language procedure **DecodeDEFG** and the values that procedure yields are used to look-up and interpolate in the multidimensional 4-D table. It is worth noting that 4-D table is mechanistically styled after the multidimensional table of the rendering table of the **colorrendering** dictionary (CRD). The processes generated by **DecodeDEFG** yield the (*A*, *B*, *C*) values. Afterwards, the (*A*, *B*, *C*) values are processed as **CIEBasedABC** values by the color space array (CSA) and passed to the CRD in the normal way as illustrated in Figure 5.4 above and as described in section 4.8.3 of the *PostScript Language Reference Manual, Second Edition*.

The dictionary for **CIEBasedDEFG** is defined as follows in Table 5.9. The dictionary for the **CIEBasedDEF** color space is the same as below except that the inputs are three components. That is, **RangeDEF** and **RangeHIJ** have six array values giving the ranges for three components; **DecodeDEF** is a three-values array; finally, the look-up table, **Table**, is a 3-input/3-output mapping table with corresponding adjustments in its parameters.

After execution of **setcolorspace**, the initial values of *D*, *E*, *F*, and *G* are 0, unless the range of valid values for a color component does not include 0, in which case the nearest valid value is substituted.

Table 5.9 *Entries in a CIEBasedDEFG color space dictionary*

Key	*Type*	*Semantics*
RangeDEFG	array	(*Optional*) Array of eight numbers [*D*0 *D*1 *E*0 *E*1 *F*0 *F*1 *G*0 *G*1] that specify the range of valid values for the *D*, *E*, *F* and *G* components of the color space — that is, $D_0<=D<=D_1$, $E_0<=E<=E_1$, and so forth. The default value equals [0 1 0 1 0 1].
DecodeDEFG	array	(*Optional*) Array of four PostScript language procedures [*DD DE DF DG*] that decode the D, E, F and G components of the color space into values *H*, *I*, *J* and *K*, respectively, that are more suitable for performing a table look-up. Default value is the array of identity procedures [{} {} {} {}].
RangeHIJK	array	(*Optional*) Array of eight numbers [*H*0 *H*1 *I*0 *I*1 *J*0 *J*1 *K*0 *K*1] that specify the range of the valid values for the *H*, *I*, *J* and *K* components of the color space. In other words, $H_0<=H<=H_1$, $I_0<=I<=I_1$, and so forth. Default value equals [0 1 0 1 0 1].
Table	array	(*Required*) Array of the form [N_H N_I N_J N_K *table*] which describes a four-dimensional look-up table that maps colors in the four-dimensional color space with coordinates *H*, *I*, *J* and *K* into a three-dimensional color space with coordinates *A*, *B* and *C* via table look-up and interpolation. The *ABC*-space subsequently maps into the CIE 1931 (XYZ)-space using the same process and guided by the same dictionary entries as in the **CIEBasedABC** color space. Those dictionary entries are in this dictionary. The table contains $N_H \times N_I \times N_J \times N_K$ entries, each of which consists of 3 values making up an *ABC* color value. N_H, N_I, N_J and N_K must be integers greater than 1. The entry in the table at coordinates (*h*, *i*, *j*, *k*), where $0<=h<N_H$, $0<=i<N_I$, and so forth, contains the color value in the *ABC*-space that corresponds to the value in the *HIJK*-space where: $H = H_0 + h[(H_1 - H_0) / (N_H - 1)]$ $I = I_0 + i[(I_1 - I_0) / (N_I - 1)]$ $J = J_0 + j[(J_1 - J_0) / (N_J - 1)]$ $K = K_0 + k[(K_1 - K_0) / (N_K - 1)]$ where H_0, H_1, I_0 and I_1 and so forth are given in the **RangeHIJK** entry. The element *table* must be an array of N_H arrays. Each of those arrays must contain an array of N_1 strings. Each of those strings must contain 3 x N_J x N_K characters. The h*th* array value of the mapping table contains an array whose i*th* value is the string for which the 3 characters starting at position 3 x (j x N_1 + k) constitute the table entry location (*h*, *i*, *j*, *k*). These 3 characters are interpreted as color values in the *ABC* color space, each in the range 0 to 255.

Other entries	entry-specific	All of the entries required for a **CIEBasedABC** dictionary are also required entries in this type of dictionary. See the **CIEBasedABC** dictionary specification for their details. All of the entries specified as optional for a **CIEBaseABC** dictionary are also optional entries in this type of dictionary. Again, see the **CIEBasedABC** dictionary specification for their details.

5.8 Fax Environment Interface

The section 5.8.1 on page 191 describes changes to the administrative resources in the fax environment; section 5.8.2 on page 199 describes changes to the translations dictionaries in the **FaxDefaultProcs Procset**.

5.8.1 Administrative Resources

Operators have been defined to assist in overseeing the general operation of the fax printer. Job records and telephone connections can be checked with these operators. The operators are available as a Level 2 resource named **FaxAdminOps** in the category **ProcSet**. They may be reached by use of the Level 2 **findresource** operator:

```
/FaxAdminOps /ProcSet findresource begin
```

In this example, a dictionary containing the administrative fax operator definitions is placed on the operand stack. The **begin** that follows moves that dictionary to the dictionary stack so that the operators can be executed directly.

Job Records

The fax printer keeps logs on various aspects of all transmitted and received fax jobs. These records can be accessed by using the four operators described below.

deletejobsforall *proc* **deletejobsforall** *bool*

Selectively deletes entries from the logs.

jobsforall *proc* **jobsforall** -

Allows programmers to perform other functions besides reports.

reportjoblist *proc* **reportjoblist** *bool*

Prints one or more pages on the printer listing the log entries. This is similar to the activity report which may be generated automatically. For more information, see the %Fax% parameter **ActivityReport** in Table 3.41 on page 143. See also section 2.3.5.

returnjoblist *proc* **returnjoblist** *bool*

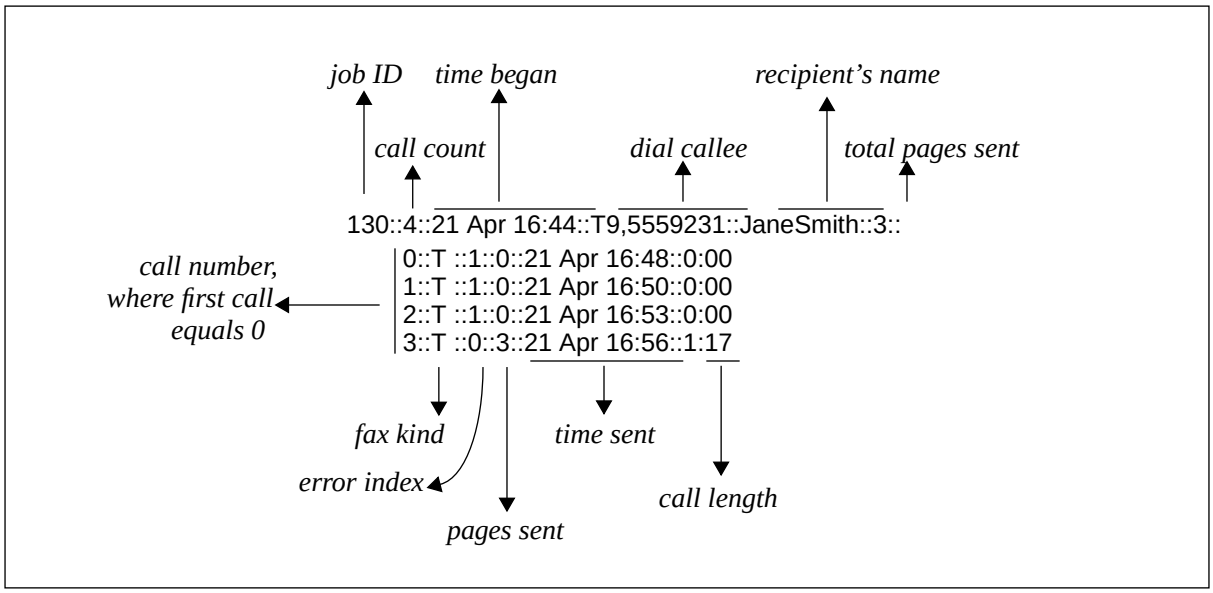
Sends an ASCII string to the connected computer made up of individual log entries. The components of each log entry are separated by double colons (::).

Each of the above operators goes through the list of log entries. For each entry, a job dictionary is placed on the operand stack, and then the procedure proc is executed. proc is expected to consume the job dictionary from the operand stack. For **deletejobsforall**, the proc must leave a boolean on the stack indicating whether the log entry should be deleted—*true* means delete. For **reportjoblist** and **returnjoblist**, the proc must leave a boolean on the stack indicating whether the entry is to be included in the report—*true* means it is included. For **jobsforall** the proc should carry out whatever actions it desires and not push a return value on the stack.

Interpreting returnjoblist Output

The following figure illustrates how to interpret the output returned by the **returnjoblist** operator.

Figure 5.6 *A sample fax log entry returned by returnjoblist*



In the figure above, the information on the first line comes from the **job** dictionary. Each line that follows summarizes information from the **calls** dictionary.

Table 5.10 *Entries in a fax job dictionary*

Entry	*Type*	*Semantics*
CallCount	integer	The value of **CallCount** indicates the number of calls involved in this transmission; it is also the number of entries in the **Calls** array. This value is always 1 for received faxes.
CalleeID	string	For received faxes, this is empty. For transmitted faxes, this is the string the called machine used to identify itself. This is the station ID which is usually the phone number or company name for the receiving machine.
Calls	array of dictionaries	The dictionaries in the **Calls** array describe the individual calls of the possibly multiple call fax job. The contents of each dictionary in **Calls** are described in Table 5.11.
DialCallee	string	See the description of **DialCallee** in Table 2.6.
EmailDest	string	For sent faxes, this is the **RecipientID** string from the **FaxOptions** dictionary. For received faxes, this is the string supplied in Adobe Non-standard Facilities frame, if any.
ErrorArray	array of strings	This is an array of strings which describe status conditions. It is indexed by the **ErrorIndex** entries described in Table 5.11.
HostJobID	integer	For transmitted faxes, the value is the same as the one for **HostJobID** in the **FaxOptions** dictionary. For faxes received, it is always 0.

JobId	integer	This entry is a unique identifier for this send or receive job.
ReceiverCapabilities	dictionary	For send jobs, this dictionary contains information about the receiver's abilities as learned during the transmission. For receive jobs, the dictionary is empty. The dictionary has eight entries for send jobs. For a detailed description of the entries in the **ReceiverCapabilities** dictionary, see Table 5.12.
RecipientLanguage	string	For transmitted faxes, the value is the same as was given for **RecipientLanguage** in the job's **FaxOptions** dictionary. If this was not specified then the value of the (%Fax%) parameter **LocalLanguage** (at the time the job was prepared) is given. For received faxes, this string is empty.
RecipientName	string	For transmitted faxes, the value of **RecipientName** is the same as was given in the job's **FaxOptions** dictionary. If **RecipientName** in the job's **FaxOptions** dictionary is *null*, a non-*null* value to store in the log is sought according to the scheme described under **RecipientName** in Table 2.6. For received faxes, this item is not present.
RecipientOrg	string	For transmitted faxes, the value of **RecipientOrg** is the same as was given in the job's **FaxOptions** dictionary. If **RecipientOrg** in the job's **FaxOptions** dictionary is *null*, a non-*null* value to store in the log is sought according to the scheme described under **RecipientOrg** in Table 2.6. For received faxes, this item is not present.
RecipientPhone	string	For transmitted faxes, the value of **RecipientPhone** is the same as was given in the job's **FaxOptions** dictionary. If **RecipientPhone** in the job's **FaxOptions** dictionary is *null*, a non-*null* value to store in the log is sought according to the scheme described under **RecipientPhone** in Table 2.6. For received faxes, this item is not present.
SenderName	string	For transmitted faxes, the value of **SenderName** is the same as was given in the job's **FaxOptions** dictionary. If **SenderName** in the job's **FaxOptions** dictionary is *null*, a non-*null* value to store in the log is sought according to the scheme described under **SenderName** in Table 2.6. For received faxes, this item is not present.
SenderOrg	string	For transmitted faxes, the value of **SenderOrg** is the same as was given in the job's **FaxOptions** dictionary. If **SenderOrg** in the job's **FaxOptions** dictionary is *null*, a non-*null* value to store in the log is sought according to the scheme described under **SenderOrg** in Table 2.6. For received faxes, this item is not present.

SubAddress	string	For sent faxes, this is the **RecipientID** string from the **FaxOptions** dictionary. For received faxes, this is the string sent by the transmitter in the subaddress frame (if any) during the initial protocol negotiations.
TimeBegan	array of integers	This parameter indicates the time when the job was submitted.
TotalPages	integer	This parameter indicates the sum of all pages imaged or received.
TotalPagesSent	integer	This parameter indicates the number of pages transmitted (not including cover sheets) or printed.

Table 5.11 details the entries in the **Calls** dictionary.

Table 5.11 *Entries in a Calls dictionary*

Entry	*Type*	*Semantics*
CallLength	integer	This key designates the length of the call in seconds.
CoverPagesSent	integer	This key designates a count of the number of cover sheets transmitted in this call; for received faxes this is always 0.
ECMused	boolean	This key indicates whether the error correcting method was used during transmission.
ErrorIndex	integer	This key designates the final status for the call. This integer can be used as an index into **ErrorArray**.
FaxKind	integer	This key designates the type of fax transmitted or received. The fax type is determined by an integer code: 0 Group 3 raster transmission. 1 PostScript language file transmission. 2 Group 3 raster reception. 3 PostScript language file reception.
Format	integer	This key shows what mode of compression was used on the Group 3 data. 0 MH compression. 1 MR compression. 2 MMR compression.
Pages	integer	This key designates the number of pages prepared for this call or received. Compare with the definition of **PagesSent**.
PagesSent	integer	This key designates the number of pages actually transmitted or printed. Compare with the definition of **Pages**.
Resolution	integer	This key indicates the resolution of the Group 3 data that was transmitted.

	0	Used standard ITU Group 3 resolution.
	1	Used fine ITU Group 3 resolution.
Speed	integer	This key indicates the transmission speed used during the call. Possible values are 2400, 4800, 7200 and 9600.
TimeSent	array of integers	This key designates the time when the call started.

Table 5.12 details the entries found in the **ReceiverCapabilities** dictionary of the fax job dictionary described in Table 5.10.

Table 5.12 *Entries in a ReceiverCapabilities dictionary*

Key	*Type*	*Semantics*
UnlimitedLength	boolean	Can do unlimited length pages.
MRCapable	boolean	Understands modified read (MR) compression.
MMRCapable	boolean	Understands modified modified read (MMR) compression.
ECMCapable	boolean	Supports error correction mode (ECM).
PSFTCapable	boolean	Supports PostScript file transmission.

tryphone

The **tryphone** operator allows diagnosis of the attachment to the telephone system. The operator takes two arguments and returns a string:

tryphone *string int* **tryphone** *string*

The *string* object is the telephone number to be dialed in the test. The telephone number must be in the same format as described under **DialCallee** in Table 2.6. The input integer *int* is a code indicating the kind of test to be performed. If *int* is 0, **tryphone** will wait until a dial tone is detected after dialing the string. If *int* is 1, **tryphone** will wait until it receives a handshake from the remote fax machine. The returned *string* indicates the results of the test. If *string* is (Heard fax machine), then the handshake with the remote fax machine was successful (and the phone connection was dropped politely with no data transmission). Other *string* values indicate the results of the test. The strings that may be returned are listed below.

(Heard fax machine.)
(Heard dial tone.)
(No dial tone.)

The **tryphone** operator will raise a PostScript language **ioerror** if fax transmission is not enabled (see **ServiceEnable** in Table 3.41 on page 143) or if the fax hardware is not properly installed.

Job Queues

Two operators in the **FaxAdminOps ProcSet** instance may be used to examine the status of incoming and outgoing faxes which are currently in process or have recently finished being sent or received. They are as follows.

transmitjobsforall *proc* **transmitjobsforall** -

This operator goes through all transmit jobs that the fax printer is aware of. For each job it composes a dictionary describing the state of that job, pushes the dictionary on the stack and calls the *proc*. The *proc* is free to do whatever it wishes, but must end up removing the dictionary from the stack. In the group of jobs that the operator enumerates will be

- all finished transmit jobs which still have records in the non-volatile log storage;
- jobs waiting to be transmitted because they have asked for delayed transmission, are waiting to retry, or simply have not had their turn yet on the phone;
- the job currently using the modem if it is a transmit job.

Refer to Table 5.13 for details on the dictionary created by **transmitjobsforall**.

receivejobsforall *proc* **receivejobsforall**

This operator goes through all received jobs that the fax printer is aware of. For each job it composes a dictionary describing the state of that job, pushes the dictionary on the stack and calls the proc. The proc is free to do whatever it wishes, but must end up removing the dictionary from the stack. In the group of jobs that the operator enumerates will be

- all finished receive jobs which still have records in the non-volatile log storage;
- jobs waiting to be printed;
- the job currently using the modem if it is a receive job.

Refer to Table 5.14 for details on the dictionary created by **receivejobsforall**.

Dictionaries are generated by the two **FaxAdminOps ProcSet** operators **receivejobsforall** and **transmitjobsforall.** The following two tables detail those dictionaries.

Table 5.13 *Dictionary generated by transmitjobsforall*

Key	*Type*	*Semantics*
HostJobID	integer	From job's **FaxOptions** options dictionary.
Status	integer	The following integers map to these meanings: 0 Finished successfully. 1 Finished with error. 2 Waiting to send. 3 Currently sending.
DialCallee	string	The dial string used to dial out.
Duration	integer	Total number of seconds on the phone spent by this job so far.
ErrorIndex	integer	Summary error code; index into **ErrorArray.**
ErrorArray	array of strings	Array of strings describing status conditions.
Kind	integer	0 for raster fax, 1 for PostScript language file transmission jobs (i.e., sending the PostScript language file).
Pages	integer	Number of pages (excluding covers) sent if raster and job is finished or number prepared if job is not finished; 1 for PostScript file transmission jobs.
ReceiverPSCapable	boolean	The value here is *true* if during the initial handshake, the receiving machine said that it was capable of receiving PostScript transmissions (whether or not that was what was being attempted at the time).
RetriesTried	integer	Number of unsuccessful calls; meaningful only for waiting or sending jobs.
RetriesLeft	integer	Number of retries left for this job; meaningful only for waiting or sending jobs.
Time	array of integers	This array has the same format and range of values as the **MailingTime** array. For finished jobs (**Status** 0 or 1) this is the time when the (last) call started. For jobs waiting to be sent, this is when the job is scheduled to be sent; this also covers retries. For jobs that are currently being sent this is the current time.

Table 5.14 details the dictionary generated by **receivejobsforall**.

Table 5.14 *Dictionary generated by receivejobsforall*

Key	*Type*	*Semantics*
Status	integer	The following integers map to these meanings: 0 Finished successfully. 1 Finished with error. 2 Waiting to be printed. 3 Currently receiving.
CallerID	string	Station ID of caller.
Duration	integer	Total number of seconds on the phone spent by this job so far.
ErrorIndex	integer	Summary error code; index into **ErrorArray.**
ErrorArray	array of strings	Array of strings describing status conditions
Kind	integer	0 for raster fax, 1 for PostScript language file transmission jobs.
Pages	integer	Number of pages received if raster, 1 if PostScript file transmission jobs.
Recipient	string or null	Subaddress field or other in-bound routing information.
Time	array of integers	Time the call started.

5.8.2 Translations Dictionaries in the FaxDefaultProcs ProcSet

As noted earlier (see section 2.2.5), a writeable **ProcSet** instance called **FaxDefaultProcs** contains the default procedures for cover sheets, transmission reports, page captions and activity reports. These procedures all generate their text by looking in a language specific dictionary of messages. The dictionary to use is selected by the procedure from a dictionary of such dictionaries based on a language key. This dictionary of translation dictionaries is contained in the **FaxDefaultProcs ProcSet** instance and can therefore be overwritten. Thus, the text of existing messages can be changed, and entirely new languages can be added. Six translation dictionaries are present initially—those for English, German, French, Spanish, Italian and Dutch. The fonts used with a particular language are determined by entries in that language's translation dictionary, and are therefore also changeable.

The dictionary of translation dictionaries located in the **FaxDefaultProcs ProcSet** instance is called **TranslationDicts**:

```
/TranslationDicts <<
  /languageName1 translationDict1
  /languageName2 translationDict2
  ...
  /languageNameN translationDictN
  >> bind def
```

Each dictionary in **TranslationDicts** contains the same set of keys. These are described below in Table 5.15.

Many of the keys are strings. Some of the strings are used to generate messages containing variable information such as the number of pages or number of phone calls. The characters %1 (and sometimes also %2 and %3) are used to indicate the points at which the variable data will be inserted. For example, the translation dictionary entry

```
(We sent %1 pages.)
```

can be used to generate messages on a report such as

```
We sent 15 pages.
```

Similarly,

```
(It took %1 calls, lasting a total of %2 seconds.)
```

could give rise to

```
It took 3 calls, lasting a total of 341 seconds.
```

Other keys in the dictionary are procedures that return a font on the stack. These procedures are called by the default report procedures to obtain the correct fonts for various uses—page titles, column heading, variable data, and so on. The report procedures scale the returned fonts to the sizes needed. The fonts returned are assumed to use the encoding vector identified by the **Encoding** resource category instance **IOSLatin1Encoding**.

Table 5.15 *Entries in a dictionary contained in TranslationDicts*

Key	*Type*	*Default value in the English dictionary*
Items used on more than one report:		
ToLabel	string	(To:).
FromLabel	string	(From:).
PagesLabel	string	(Pages).
FaxNumberLabel	string	(Fax Number).

PageLabel string (Page).

NameLabel string (Name).

ExcludingCovers string (%1, excluding cover sheets.).

Items used by the cover page procedure:

CoverSheetTitle string (Facsimile Cover Sheet).

BackCoverTitle string (Facsimile Back Cover).

FollowingPages string (Please forward the following pages to:).

PrecedingPages string (Please forward the preceding pages to:).

RecipientLabel string (Recipient).

OrganizationLabel string (Organization).

PhoneNumberLabel string (Phone Number).

MailStopLabel string (Mail Stop).

SenderLabel string (Sender).

DateLabel string (Date).

SubjectLabel string (Subject).

NoteLabel string (Note).

WhomIt string (Whom it may concern).

Unspecified string (Unspecified).

PSTransDesc string (--- PostScript language Transmission ---).

UnknownDueTo string (PostScript language Transmission).

Items use by the confirmation report procedure:

TransRepHeader string (Transmission Status Report).

Successful string (The following transmission completed successfully.).

NotSuccessful string (The following transmission was not successful.).

LocationLabel string (Location:).

CallerIDLabel string (Caller ID:).

CalleeIDLabel string (Callee ID:).

PrinterNameLabel string (Printer name:).

PagesUnknownPS string (PostScript language transmission.).

MailedLabel string (Mailed:).

StatusLabel string (Status:).

DurationLabel string (Duration:).

SingleCallOneMinAndSecs
string (1 minute and %1 seconds.).

SingleCallMinsAndOneSec
string (%1 minutes and 1 second.).

SingleCallOneMinAndOneSec
string (1 minute and 1 second.).

SingleCallMinsAndSecs
string (%1 minutes and %2 seconds.).

SingleCallSecondsOnly
string (%1 seconds.).

MultiCallOneMinAndSecs
string (1 minute and %1 seconds in %3 calls.).

MultiCallMinsAndOneSec
string (%1 minutes and 1 second in %3 calls.).

MultiCallOneMinAndOneSec
string (1 minute and 1 second in %3 calls.).

MultiCallMinsAndSecs
string (%1 minutes and %2 seconds in %3 calls.).

MultiCallSecondsOnly
string (%1 seconds in %3 calls.).

Items used by the activity report procedure:

ActRepHeader string (FAX ACTIVITY REPORT).

StartTimeLabel string (Start Time).

ActivityLabel string (Activity).

CallDurationLabel string (Call Duration).

PhoneNumberLabel string (Phone Number).

PagesSentOrPrintedLabel
string (Pages Sent or Printed).

ExplActCodes string (Explanation of Activity codes:).

TransGroup3 string (Transmit group 3).

TransPS string (Transmit PostScript).

ReceiveGroup3 string (Receive group 3).

ReceivePS string (Receive PostScript).

FaxKindAbbrev array [(T)(TPS)(R)(RPS)].

Items used to describe groups of pages:

ExcludingCover string (%1, excluding cover sheet.).

ThisFinalGroup string (%1 this final group, %2 entire document.).

ThisGroupExcludingCovers
string (%1 this group, excluding cover sheet(s).).

SeeTrailers string (See trailer sheet(s) to follow.).

AlreadySent string (%1 already sent, trailer sheet(s) to follow.).

NoPagesSent string (Fax - No pages sent to destination.).

Error messages:

ErrInRecvPS string (Error in received PS code - job aborted).

OutOfMemForRecv string (Out of memory for receive - page incomplete).

ErrInCovSheetProc string (Error in Cover Sheet Procedure).

ErrInPageCapProc string (Error in Page Caption Procedure).

ErrInConfirmProc string (Error in Confirmation Procedure).

faxerrarray array

[
(No problem.)
(Callee didn't respond.)
(High-speed data transmission failed.)
(Transmission error, disconnected.)
(Possible success, but imperfect confirmation.
(Callee rejected PostScript language transmission.)
(Error in PostScript language file.)
(No dial tone.)
(Internal PS fax error.)
(Callee wasn't a PostScript language server.)
(There were no pages to send.)
(Storage to hold incoming data was exhausted.)
(Storage to assemble PostScript language transmission exhausted.)
(PS transmission rejected; Error during revert.)
(Callee number busy)
(Callee number blacklisted)
].

Items used to generate dates:

smonths array [(Jan)(Feb)(Mar)(Apr)(May)(Jun)(Jul)(Aug)(Sep)(Oct)(Nov)(Dec)].

sweekdays array [(Sun)(Mon)(Tue)(Wed)(Thu)(Fri)(Sat)(Sun)].

InvalidTime string (<<Invalid time>>).

Procedures returning fonts:

ActRepTitleFont procedure Procedure returning Helvetica with **ISOLatin1Encoding**.

ActRepHeadingsFont
procedure Procedure returning Helvetica with **ISOLatin1Encoding**.

ActReportDataFont
procedure Procedure returning Courier with **ISOLatin1Encoding**.

ConfRepTitleFont
procedure Procedure returning Times-Bold with **ISOLatin1Encoding**.

ConfRepLabelFont
procedure Procedure returning Times-Bold with **ISOLatin1Encoding**.

ConfRepDataFont
procedure Procedure returning Times-Roman with **ISOLatin1Encoding**.

ConfRepSummary
procedure Procedure returning Times-Italic with **ISOLatin1Encoding**.

CaptionFont	procedure	Procedure returning Courier with **ISOLatin1Encoding**.
CoverLabelFont	procedure	Procedure returning Helvetica-Bold with **ISOLatin1Encoding**.
CoverDataFont	procedure	Procedure returning Bookman-Light with **ISOLatin1Encoding**.

CHAPTER 6

Compatibility

The PostScript language has undergone several significant extensions. It is designed to be a universal standard for device-independent page descriptions, but each PostScript language implementation supports features and capabilities particular to that implementation. Appendix D, “Compatibility Strategies,” in the *PostScript Language Reference Manual, Second Edition*, presents guidelines for taking advantage of language extensions while maintaining compatibility with all PostScript interpreters.

Level 1 implementations provide a collection of device control and system parameter configuration operators and procedures, most of which are defined in the dictionary **statusdict**. The contents of **statusdict** are product-dependent, although an attempt has been made to maintain a consistent specification for common features. It is the dictionary for product-specific operators and other definitions.

Device control and configuration of system parameters in PostScript language Level 2 is accomplished in a standard way in the language through the device setup and interpreter parameter operators. However, for compatibility with existing Level 1 PostScript language driver software, which might depend on **statusdict** operators and keys that were often present in Level 1 PostScript products, a collection of **statusdict** operators and keys is included in each Level 2 PostScript language implementation.

Almost all of these functions are implemented as PostScript language procedures that call appropriate Level 2 operators such as **setpagedevice**.

Adobe recommends that you do not use the **statusdict** operators and keys in Level 2 PostScript language drivers because the presence or absence of the operators and keys is product-dependent. Instead, the appropriate Level 2 standard operators should be used.

6.1 Compatibility Operators

The following is a list of the compatibility operators described in this chapter. The compatibility operators are grouped by dictionary.

In the following list, as well as in the tables in this chapter, these symbols are used:

† means that this compatibility operator is typically present in all releases up to and including the 2015 PostScript language implementations.

‡ means that this compatibility operator is typically present in all releases up to and including the 2015 PostScript language implementations. However, in the absence of the associated feature, it performs no function aside from its documented effect on the operand stack.

ø means that this compatibility operator is typically present in all releases up to and including the 2015 PostScript imagesetter implementations.

§ means that this compatibility operator requires execution in a system administrator job.

¶ means that this compatibility operator can affect page device parameters.

Operators without a symbol are associated with a particular feature and are defined only if the feature is present in the product.

In **statusdict**:

a3tray¶	**a4tray**¶	**accuratescreens**ø
appletalktype	**b5tray**¶	**buildtime**†
byteorder†	**checkpassword**†	**checkscreen**ø
defaulttimeouts‡	**diskonline**	**diskstatus**
doprinterrors	**dostartpage**	**dosysstart**
duplexmode	**emulate**	**firstside**
hardwareiomode‡	**initializedisk**§	**jobname**†
jobtimeout†	**ledgertray**¶	**legaltray**¶
lettertray†¶	**manualfeed**	**manualfeedtimeout**
margins‡	**mirrorprint**	**newsheet**
pagecount‡	**pagemarginø**	**pageparamsø**
pagestackorder	**printername**†	**processcolors**
product†	**ramsize**	**realformat**†
resolution	**revision**†	**sccbatch**
sccinteractive‡	**setaccuratescreensø**	**setdefaulttimeouts**†§¶
setdoprinterrors§	**setdostartpage** §	**setdosysstart**§
setduplexmode¶	**sethardwareiomode**‡§	**setjobtimeout**‡
setmargins‡§¶	**setmirrorprint**¶	**setpage**ø¶
setpagemarginø¶	**setpageparams**ø¶	**setpagestackorder** §¶
setprintername‡§	**setresolution**¶	**setsccbatch**§
setsccinteractive‡§	**setsoftwareiomode**‡§	**settumble**¶
setuserdiskpercent§	**softwareiomode**‡	**tumble**
userdiskpercent	**waittimeout**‡	**11x17tray** ¶

In **userdict**:

a3[¶]	**a4**[¶]	**a4small**[¶]
b5[¶]	**ledger**[¶]	**legal**[‡¶]
letter[‡¶]	**lettersmall**[¶]	**note**[¶]
11x17[¶]		

In **systemdict**:

devdismount[†§]	**devforall**[†]	**devformat**[†§]
devmount[†§]	**devstatus**[†]	

6.2 Compatibility Operator Descriptions

This section describes the Level 1 compatibility objects present in Level 2 PostScript interpreters. The majority of these Level 1 objects are operators in **statusdict**. Other dictionaries may also contain compatibility objects (for example, **letter** in **userdict**). Compatibility objects need not always be operators (for example, the **waittimeout** integer in **statusdict**).

There is a Level 2 method of performing most Level 1 compatibility operations. For the following compatibility operators, there is currently no PostScript language Level 2 equivalent:

checkpassword[†]	**checkscreen**[Ø]	**devforall**[†]
emulate	**firstside**	**newsheet**
sccinteractive[‡]	**setpapertray**[¶]	**setsccinteractive**[‡§]
setuserdiskpercent[§]	**userdiskpercent**	

The remaining compatibility objects are described below in terms of Level 2 operations. This not only provides the most accurate description of the compatibility operation but also indicates the correct Level 2 method of carrying out the operation.

Because many of the compatibility operations originally dealt with product-specific behavior, the semantics of some operations in Level 1 varied from one product to another. Defining compatibility operations in terms of product-independent Level 2 operations corrects this problem at the cost of sometimes providing an imperfect emulation of the Level 1 operation.

Some Level 1 operations are no longer relevant for PostScript language Level 2 programs. In these cases, the compatibility operations may be implemented as no-ops that allow the PostScript language Level 1 program containing them to continue without generating errors. An example of such an operator is **setsccinteractive**.

6.2.1 Error Behavior

In general, the behavior for error conditions is different between the Level 1 compatibility operation and the corresponding Level 2 method. This is to provide error behavior that is as similar to Level 1 error behavior as possible. As an example, a Level 1 paper tray operation such as **lettertray** may generate a **rangecheck** while the corresponding Level 2 operation will generate a **configurationerror** or will perform other actions under the control of **Policies** in the page device dictionary.

6.2.2 Using a Password to Change Persistent Values

In Level 1, many of the operations that changed persistent values could only be executed from jobs that had "exited the server" (this action required a password). If such an operation was executed without exiting the server an **invalidaccess** error resulted.

In Level 2, the notion of exiting the server has been replaced by the concept of an unencapsulated job (see section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*). An unencapsulated job is entered by executing the Level 2 operator, **startjob**, or the Level 1 operator, **exitserver**. These operators require a password to be presented. The password must be equal to the value of either the **StartJobPassword** or the **SystemParamsPassword** system parameter. If the password is equal to the value of **StartJobPassword**, an ordinary unencapsulated job is started (see section 3.7.7 of the *PostScript Language Reference Manual, Second Edition*). If the password is equal to the value of **SystemParamsPassword**, a system administrator job is started. (If the **SystemParamsPassword** is a zero-length string or has never been set, every unencapsulated job is a system administrator job.)

Many compatibility operators change system or device parameters. Such operators use the Level 2 **setsystemparams** or **setdevparams** operators to emulate the Level 1 functionality. Those operators ordinarily require a **Password** parameter to be presented on each execution. This requirement is relaxed during a system administrator job, but not during an ordinary unencapsulated job. Since the compatibility operators do not present a password, this means they can be successfully executed only during a system administrator job. Executing them during an ordinary unencapsulated job (or any encapsulated job) will cause an **invalidaccess** error.

Compatibility operators that affect page device parameters save their persistent values only if they are executed from an unencapsulated job. In encapsulated jobs the values set by these compatibility operators will obey the normal save-restore rules and are not saved to persistent storage.

Note *The compatibility objects are present in Level 2 printers for compatibility purposes only, and their use in PostScript language Level 2 programs is strongly discouraged.*

statusdict Compatibility Operators

a3tray¶ See section 6.2.5, “Paper Tray Operations.”

a4tray¶ See section 6.2.5, “Paper Tray Operations.”

accuratescreens ø See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

appletalktype – **appletalktype** *string*

Returns a string with the same value as the **LocalTalkType** device parameter in the %LocalTalk% parameter set and the **EtherTalkType** parameter in the %EtherTalk% parameter set. Redefining **appletalktype** will cause the **LocalTalkType** parameter to change as well as the **EtherTalkType** parameter. Similarly, changes to the **EtherTalkType** or the **LocalTalkType** parameter will change the string returned by the **appletalktype** operator.

The compatibility operator **appletalktype** is present only if either the %LocalTalk% or %EtherTalk% device name is present.

Errors: **stackoverflow**

b5tray¶ See section 6.2.5, “Paper Tray Operations.”

buildtime † – **buildtime** *int*

Returns an integer with the same value as the system parameter **BuildTime**.

Errors: **stackoverflow**

byteorder † – **byteorder** *bool*

Returns a boolean with the same value as the system parameter **ByteOrder**.

Errors: **stackoverflow**

checkpassword †

int **checkpassword** *bool*
string **checkpassword** *bool*

Checks whether *string* or *int* (*int* is converted to a string) is a valid password for either **SystemParamsPassword** or **StartJobPassword**. If valid, *true* is returned, otherwise *false* is returned. If either password is not set, then *true* will be returned. A returned value of *true* indicates that *string* or *int* is a valid argument to **startjob** and **exitserver**. There is no PostScript language Level 2 equivalent for **checkpassword**.

Errors: **stackunderflow**, **typecheck**

checkscreen ø

See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

defaulttimeouts ‡

– **defaulttimeouts** *job manualfeed wait*

Returns the values of the system parameters **JobTimeout** and **WaitTimeout** and the page device parameter **ManualFeedTimeout** for *job*, *wait* and *manualfeed*, respectively. **defaulttimeouts** always returns three values, even if the corresponding system parameters are not present (zeros are returned in this case).

Errors: **stackoverflow**

diskonline

– **diskonline** *bool*

Returns *true* if and only if a writeable disk device is mounted. This is determined by searching all device parameter sets named %disk*%, where * represents zero or more additional characters in the name. If the **Writeable** parameter is *true* for any of the sets searched, *bool* is set to *true*; otherwise, it is set to *false*. Note that a disk parameter set with **Writeable** *true* need not have an initialized file system.

Errors: **stackoverflow**

diskstatus

– **diskstatus** *free total*

Returns the number of disk pages (a page is 1024 characters) free and the total number of pages available on all writeable disk devices. This is determined by searching all device parameter sets named %disk*% that have a **Writeable** parameter set to *true*. The * represents zero or more

additional characters in the name. *free* is the sum of the **Free** parameters from all such parameter sets, and *total* is the sum of the **LogicalSize** parameters from all such parameter sets.

Errors: **stackoverflow**

doprinterrors – **doprinterrors** *bool*

Returns the value of the system parameter *DoPrintErrors*.

The system parameter *DoPrintErrors* must be present for the operator **doprinterrors** to be present.

Errors: **stackoverflow**

dostartpage – **dostartpage** *bool*

Returns the value of the system parameter **DoStartPage**.

The system parameter **DoStartPage** must be present for the compatibility operator **dostartpage** to be present.

Errors: **stackoverflow**

dosysstart – **dosysstart** *bool*

Returns *false* if and only if the value of the system parameter **StartupMode** is 0.

The system parameter **StartupMode** must be present for the compatibility operator **dosysstart** to be present.

Errors: **stackoverflow**

duplexmode See section 6.2.6, “Page Duplex Compatibility Operators.”

emulate *input-stream emulation-name* **emulate** –
or
input-stream params-dict emulation-name **emulate** –

Causes the PostScript interpreter to yield control, and the emulator named by emulation-name to start processing. The **emulate** operator is present in **statusdict** and only in products that have one or more emulators coresident with the PostScript interpreter. The exact semantics of the emulators are product-dependent and may be different in different products even though the emulation name may be the same. The specifics of each product's emulators (if any) are documented in the product *Addendum*. In most coresident emulations, the command sequence ESC-DEL-0 can be used to make the emulator yield control back to the PostScript interpreter; however, the PostScript language context will generally have been lost.

The allowed values of *emulation-name* may be found in the implicit resource category **Emulator**. An illegal *emulation-name* will cause a **rangecheck** error.

A *params-dict* argument is optional. If the named emulator does not need parameters and a *params-dict* is provided, the dictionary will be ignored. If the named emulator requires parameters and no *params-dict* is provided, then product-dependent defaults will be used if possible. Currently, no emulators require parameters.

The *input-stream* is a file object which becomes the input source for the emulator. The *input-stream* specified must be appropriate to the product-dependent emulator, as defined in the product *Addendum*. An illegal *inputstream* will cause an **invalidaccess** error.

Errors: **invalidaccess**, **rangecheck**, **stackoverflow**, **stackunderflow**

firstside See section 6.2.6, "Page Duplex Compatibility Operators."

hardwareiomode ‡ – **hardwareiomode** *int*

Returns an *int* which indicates the current communication channel whose corresponding device parameter set **Enabled** boolean is *true*. It will always return the channel indicated by **CurInputDevice** if that channel is on and enabled and one of the ones listed below. Otherwise, the smallest such int is returned. If none in the list is on and enabled, 0 is returned. The interpretation of *int* is:

0	%Serial%
1	%Parallel%
2	%LocalTalk%
3	%SerialB%

The **Serial**, **Parallel**, **SerialB**, or **LocalTalk** device parameter set must be present for the compatibility operator **hardwareiomode** to be present.

Errors: **stackoverflow**

initializedisk § *pages action* **initializedisk** –

Initializes each writeable disk, setting the disk device parameters **LogicalSize** and **InitializeAction** to the value of *pages* and *action*+1, respectively.

Errors: **invalidaccess**, **ioerror, rangecheck**, **stackunderflow**, **typecheck**

jobname † – **jobname** *string*

A string with the same value as the user parameter **JobName**. Redefining either **jobname** or the user parameter **JobName** redefines the other to the same value.

The user parameter **JobName** must be present for the compatibility operator **jobname** to be present.

Errors: **stackoverflow**

jobtimeout † – **jobtimeout** *int*

Returns the value of the user parameter **JobTimeout**.

Errors: **stackoverflow**

ledgertray¶ See section 6.2.5, “Paper Tray Operations.”

legaltray¶ See section 6.2.5, “Paper Tray Operations.”

lettertray†¶ See section 6.2.5, “Paper Tray Operations.”

manualfeed – **manualfeed** *bool*

A boolean that works in conjunction with the page device parameter **ManualFeed** to determine whether a page is fed manually. If either **manualfeed** or **ManualFeed** is *true* at the time of a **showpage** or **copypage**, then that page will be fed manually; otherwise, the page will not be fed manually.

The values of **ManualFeed** and **manualfeed** are determined independently. That is, setting the **manualfeed** boolean or setting the **pagedevice** parameter, **ManualFeed**, does not affect the value of the other.

The **manualfeed** key is present in **statusdict** if and only if the page device parameter **ManualFeed** is defined for the product. The initial value of **manualfeed** at power-on is *false*.

Errors: **stackoverflow**

manualfeedtimeout – **manualfeedtimeout** *int*

An integer that works in conjunction with the page device parameter **ManualFeedTimeout** to determine the **manualfeed** time-out for any given page. By default, **manualfeedtimeout** is not defined in **statusdict,** and in that case the value of the page device parameter **ManualFeedTimeout** is used to determine the time-out value. If a job has defined **manualfeedtimeout** to be an integer value in **statusdict,** then this value will be used instead of **ManualFeedTimeout** for the time-out value.

The values of **ManualFeedTimeout** and **manualfeedtimeout** are determined independently. That is, setting the **manualfeedtimeout** integer or setting the page device parameter **ManualFeedTimeout** does not affect the value of the other.

Errors: **stackoverflow**

margins‡ – **margins** *top left*

Returns the *x* and *y* components of the page device parameter **Margins** as *left* and *top*, respectively.

Errors: **stackoverflow**

mirrorprint – **mirrorprint** *boolean*

Returns the value of the page device parameter **MirrorPrint**.

Errors: None.

newsheet See section 6.2.6, “Page Duplex Compatibility Operators.”

pagecount ‡ – **pagecount** *int*

Returns the value of the system parameter **PageCount**.

Errors: **stackoverflow**

pagemargin ø See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

pageparams ø See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

pagestackorder – **pagestackorder** *bool*

Returns the logical complement of the page device **OutputFaceUp** boolean parameter. For example, if **OutputFaceUp** is *true*, *bool* will be *false*.

The page device parameter **OutputFaceUp** must be present for the compatibility operator **pagestackorder** to be present.

Errors: **stackoverflow**

printername † *string* **printername** *substring*

Stores the value of the system parameter **PrinterName** in *string* and returns a string object designating the *substring* actually used.

Errors: **rangecheck, stackunderflow**, **typecheck**

processcolors – **processcolors** *int*

Returns the number of device process color components in the current page device (1 for black, 3 for RGB or CMY, or 4 for CMYK). The **statusdict** compatibility operator **processcolors** is mandatory on products that can produce more than one color but is optional on monochrome products. Traditionally, this compatibility operator does not appear on monochrome printers. Its absence indicates a monochrome-only device (1 process color).

Errors: **stackoverflow**

product † – **product** *string*

A string in **statusdict** initialized to the value of the string **product** in **systemdict**.

Errors: **stackoverflow**

ramsize – **ramsize** *int*

Returns the number of bytes of RAM available to the product. Refer to the **RamSize** system parameter.

Errors: **stackoverflow**

realformat † – **realformat** *string*

A string with the same value as the system parameter **RealFormat**.

Errors: **stackoverflow**

resolution – **resolution** *bitsperinch*

Returns the first component of the **HWResolution** array for the current output device.

Errors: **stackoverflow**

revision † – **revision** *int*

An integer with the same value as the system parameter **Revision**.

Errors: **stackoverflow**

sccbatch See section 6.2.3, “SCC Operations.”

sccinteractive ‡ See section 6.2.3, “SCC Operations.”

setaccuratescreens ø See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

setdefaulttimeouts †§¶ *job manualfeed wait* **setdefaulttimeouts** –

This compatibility operator sets the system parameters **JobTimeout** and **WaitTimeout** to *job* and *wait*, respectively, and sets the page device parameter **ManualFeedTimeout** to *manualfeed*. **setdefaulttimeouts** always takes three values, even if the corresponding system or page device parameters are not present.

Errors: **invalidaccess**, **rangecheck, stackunderflow**, **typecheck**

setdostartpage § *bool* **setdostartpage** –

This compatibility operator sets the system parameter *DoStartPage* to the value of *bool*. The system parameter *DoStartPage* must be present for the compatibility operator **setdostartpage** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setdoprinterrors § *bool* **setdoprinterrors** –

This compatibility operator sets the system parameter **DoStartPage** to the value of *bool*.

The system parameter **DoStartPage** must be present for the compatibility operator **setdostartpage** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setdosysstart [§] *bool* **setdosysstart** –

This compatibility operator sets the system parameter **StartupMode** according to the value of bool. **StartupMode** is set to 1 if *bool* is *true* and set to 0 if *bool* is *false*.

The system parameter **StartupMode** must be present for the compatibility operator **setdosysstart** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

sethardwareiomode [‡§] *int* **sethardwareiomode**

Opens specified channel(s) for communications and closes all other channels. The variable *int* specifies which communication channel(s) should be opened by setting the **On** and **Enabled** device parameters to *true*. All other channels will be explicitly closed by setting the **On** and **Enabled** parameter to *false*. The interpretation of *int* is:

0	Open %Serial% and %SerialB%. Close all others.
1	Open %Parallel%. Close all others.
2	Open %LocalTalk% and %EtherTalk% (if both exist). Close all others.
	Open %LocalTalk% (if only %LocalTalk% exists). Close all others.
	Open %EtherTalk% (if only %EtherTalk% exists). Close all others.
3	Open %Serial% and %SerialB%. Close all others.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setjobtimeout [‡] *int* **setjobtimeout** –

This compatibility operator sets the user parameter **JobTimeout** to the value of *int*.

The user parameter **JobTimeout** must be present for the compatibility operator **setjobtimeout** to be present.

Errors: **stackunderflow**, **typecheck**

setmargins ‡§¶ *top left* **setmargins** –

This compatibility operator sets the page device **Margins** parameter to [*left top*].

The page device parameter **Margins** must be present for the compatibility operator **setmargins** to be present.

Errors: **invalidaccess, rangecheck, stackunderflow**, **typecheck**

setmirrorprint¶ *boolean* **setmirrorprint** –

Creates a new page device with the parameter **MirrorPrint** set to *boolean*.

Errors: **stackunderflow**, **typecheck**

setpage ø¶ See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

setpagemargin ø¶ See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

setpageparams ø¶ See section 6.2.8, “Imagesetter Compatibility Operators Found in Statusdict.”

setpagestackorder §¶ *bool* **setpagestackorder** –

This compatibility operator sets the page device **OutputFaceUp** parameter to the logical complement of *bool*. For example, if *bool* is *true* **OutputFaceUp** is set to *false*.

The page device parameter **OutputFaceUp** must be present for the compatibility operator **setpagestackorder** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setprintername ‡§ *string* **setprintername** –

This compatibility operator sets the system parameter **PrinterName** to the value of *string*.

The system parameter **PrinterName** must be present for the compatibility operator **setprintername** to be present.

Errors: **invalidaccess**, **limitcheck**, **stackunderflow**, **typecheck**

setresolution¶ *bitsperinch* **setresolution** –

Creates a new page device with the parameter **HWResolution** set to [*bitsperinch bitsperinch*].

Errors: **rangecheck**, **stackunderflow**, **typecheck**

sccbatch See section 6.2.3, “SCC Operations.”

sccinteractive ‡ See section 6.2.3, “SCC Operations.”

setsoftwareiomode ‡§ *int* **setsoftwareiomode** –

This compatibility operator sets the values of the **Interpreter**, and if appropriate, **Protocol** device parameters for the current communications device parameter set (as indicated by the system parameter **CurInputDevice**). The meaning of *int* is:

int	*Interpreter value*	*Protocol value*
0	/PostScript	/Normal
1	/ProprinterXL	/Raw
2	/Diablo630	/Raw
3	(Reserved.)	
4	/HP7475A	/Raw
5	/LaserJetIIP (If the LaserJet IIP emulator is present in the product.)	/Raw
5	/LaserJetIII (If the LaserJet III emulator is present in the product.)	/Raw
100	/PostScript	/Binary

Note *A product will probably never have both the LaserJet IIP and LaserJet III emulators installed. If a product does have both emulators installed, passing a value of 5 to setsoftwareiomode will select only LaserJet IIP.*

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

settumble¶ See section 6.2.6, “Page Duplex Compatibility Operators.”

setuserdiskpercent § *int* **setuserdiskpercent** –

Pops *int* off the stack. This operator is essentially a no-operation instruction.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

softwareiomode ‡ – **softwareiomode** *int*

Returns *int*, which indicates (see **setsoftwareiomode**) the interpretation mode for the current communications device (as indicated by the system parameter **CurInputDevice**).

Note *If the Interpreter is not one of the values that can be set via setsoftwareiomode, softwareiomode will return -1.*

Errors: **stackoverflow**

tumble See section 6.2.6, “Page Duplex Compatibility Operators.”

userdiskpercent – **userdiskpercent** *int*

Returns the value 0. This operator is essentially a no-operation instruction.

Errors: **stackoverflow**

waittimeout ‡ – **waittimeout** *int*

Is an integer with the same value as the user parameter **WaitTimeout**. Redefining either **waittimeout** or the user parameter **WaitTimeout** redefines the other to the same value.

The user parameter **WaitTimeout** must be present for the compatibility operator **waittimeout** to be present.

Errors: **stackoverflow**

11x17tray¶ See section 6.2.5, “Paper Tray Operations.”

6.2.3 SCC Operations

The SCC (Serial Communications Controller) operators use a byte options argument (an integer parameter with values in the range 0 - 255) that holds an encoding of four SCC parameters: stop bits, data bits, flow control and parity. The byte is encoded as described in Table 6.1 through Table 6.4 (bit positions 7 - 0, with 7 the high bit and 0 the low bit):

Table 6.1 *Stop bits*

Position 7	*Stop bits*
0	1 stop bit
1	2 stop bits

Table 6.2 *Data bits*

Positions 6 and 5	*Data bits*
0	Standard
1	7 bits
2	8 bits

Table 6.3 *Flow control*

positions 4, 3 and 2	*Flow control*
0	Xon/Xoff
1	Dtr
2	Etx/Ack

Table 6.4 *Parity*

Positions 1 and 0	*Parity*
0	Space
1	Odd

2	Even
3	Mark

In Level 1, the data bits and parity interacted in a non-orthogonal manner to produce a table of possible choices for data and parity that included many common desired methods of sending data. The “standard” data bits setting was only present for backward compatibility purposes with earlier versions of the SCC operators. In particular, a standard data bit setting could always be achieved with either a 7- or 8-bit data setting. In Level 2, there are analogous entries as above for the %Serial% and %SerialB% device parameter sets.

The mapping between Level 1 stop bits and flow control and Level 2 %Serial% device parameters **StopBits** and **FlowControl**, respectively, is straightforward and obvious. It is not possible to provide such a one to one correspondence between the Level 1 notion of data bits and parity and the Level 2 %Serial% device parameters **DataBits** and **Parity**. Tables 6.5 and 6.6 show the conversion between Level 1 data bits and parity and Level 2 **DataBits** and **Parity**. Notice that in going from **DataBits** and **Parity** to data bits and parity, standard parity is never used.

Table 6.5 *Options byte to device parameters conversion*

data bits & parity —> DataBits & Parity	
standard space	7 bits /space
standard mark	8 bits /none
standard odd	7 bits /odd
standard even	7 bits /even
7 bits space	7 bits /space
7 bits mark	7 bits /mark
7 bits odd	7 bits /odd
7 bits even	7 bits /even
8 bits space	8 bits /none
8 bits mark	8 bits /none
8 bits odd	8 bits /odd
8 bits even	8 bits /even

Table 6.6 *Device parameters to options byte conversion*

DataBits & Parity —>data bits & parity	
7 bits /none	7 bits mark
7 bits /space	7 bits space
7 bits /mark	7 bits mark
7 bits /odd	7 bits odd
7 bits /even	7 bits even
8 bits /none	8 bits mark
8 bits /space	8 bits space
8 bits /mark	8 bits mark
8 bits /odd	8 bits odd
8 bits /even	8 bits even

These tables are defined to provide the best compatibility with Level 1 behavior. In several cases, no correct choice is possible. For example, in Level 1 there was no support for 7 data bits with no parity (that is, the total number of data and parity bits is 7). The Level 2 setting of 7 bits /None is imperfectly mapped to 7 bits mark. Most serial hardware does not support 8-bit mark or space and for this reason these values are never generated in mapping from Level 1 to Level 2. In fact, in Level 1, 8 bits mark and space actually provided the equivalent of the Level 2 8 bits /None capability.

SCC Compatibility Operators

sccbatch *channel* **sccbatch** *baud options*

Returns the serial communications device parameter settings. The values are from either the %SerialB_NV% (if channel equals 9) or the %Serial_NV% (if channel equals 25) parameter set. The value of options is encoded as described above, and the values for data bits and parity are determined by Table 6.6. The values for baud, stop bits, and flow control are determined from the corresponding settings for the **Baud**, **StopBits** and **FlowControl** %Serial% device parameters, respectively.

Note *If the FlowControl parameter is set to /DtrLow, sccbatch will return 1 in bit position 4, 3 and 2. If the FlowControl parameter is set to something other than /XonXoff, /Dtr, /DtrLow, or /EtxAck, sccbatch will return 0 in bit position 4, 3 and 2.*

The %Serial_NV% or %SerialB_NV% device parameter set must be present for the compatibility operator **sccbatch** to be present.

Errors: **rangecheck**, **stackoverflow**, **stackunderflow**, **typecheck**

sccinteractive ‡ *channel* **sccinteractive** *baud options*

Pops the input argument off the stack and pushes 0 0 on the stack. This operator is essentially a no-operation instruction.

Errors: **invalidaccess**, **rangecheck**, **stackoverflow**, **stackunderflow**, **typecheck**

setsccbatch § *channel baud options* **setsccbatch** –

This compatibility operator sets the communications device parameters for serial communications. Either the %SerialB_NV% (if *channel* equals 9) or the %Serial_NV% (if *channel* equals 25) settings are affected. The following device parameters are affected by *baud* and *options*: **Baud**, **StopBits**, **DataBits**, **FlowControl**, **Parity**, and **CheckParity**. **Baud**, **StopBits**, and **FlowControl** are set according to the corresponding values for baud, stop bits and flow control in the *options* argument. **DataBits** and **Parity** are set based on Table 6.6 above. **CheckParity** is set according to the new **Parity** setting:

- *true* if the setting is /Odd or /Even.
- *false* if the setting is /Space or /Mark.
- Not changed if the setting is /None (parity checking is not done if **Parity** is /None independent of the setting of **CheckParity**).

The %Serial_NV% or %SerialB_NV% device parameter set must be present for the compatibility operator **setsccbatch** to be present.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setsccinteractive ‡§ *channel baud options* **setsccinteractive** –

Pops the three input arguments off the stack. This operator is essentially a no-operation instruction.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

6.2.4 Paper Size Operations

All the operators in this section are in **userdict**. Each operator executes **setpagedevice** to request a specific paper size. The only difference among these operations is the size of paper requested and the **ImagingBBox**. The "***small**" operators specify a non-*null* **ImagingBBox** while other operators specify a *null* **ImagingBBox**. These operators use the specified size as indicated below as a page device **PageSize** parameter. In addition, all these operators set the **PageSize Policy** to 7, which guarantees that the imaging area established is the requested size regardless of the medium's actual size and turns off the normal PostScript language Level 2 media matching mechanism. (For a detailed description of **PageSize Policy** 7 see Table 2.2 on page 17.) The only error that is generated is a **limitcheck** caused by insufficient memory for the requested imaging area. In Table 6.7, default units (1/72 inch) are used as the units for the **PageSize** and **ImagingBBox**.

Table 6.7 *Paper size compatibility operators (in userdict)*

Operator	*PageSize*	*ImagingBBox*
letter ‡¶	[612 792]	null
lettersmall¶	[612 792]	[25 25 587 767]
legal ‡¶	[612 1008]	null
ledger ¶	[1224 792]	null
11x17¶	[792 1224]	null
a4 ¶	[595 842]	null
a3 ¶	[842 1191]	null
a4small¶	[595 842]	[25 25 570 817]
b5 ¶	[516 729] or [499 709]	null
note ¶	[width height]	[25 25 width-25 height-25]

The **note** compatibility operator will be present only if the size [*width height*] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **letter** and **lettersmall** compatibility operators will be present only if the size [612 792] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **legal** compatibility operator will be present only if the size [612 1008] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **a4** and **a4small** compatibility operators will be present only if the size [595 842] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **b5** compatibility operator will be present only if the size [516 729] or the size [499 709] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

6.2.5 Paper Tray Operations

All of the operators in this section are in **statusdict**. Each operator executes **setpagedevice** to request a tray containing a specific paper size. The only difference among these operations is the size of paper requested. The **PageSize** requested is the same as for the corresponding page size operator discussed in the previous section and the **ImagingBBox** requested is always *null*. These operators use the specified size as indicated in Table 6.8 as a page device **PageSize** parameter.

All of these operators set the **PageSize Policy** to 0, which guarantees that a **configurationerror** is generated if a tray containing the requested paper size is not present. The implementation of the compatibility operators convert any such **configurationerror** to a **rangecheck** for compatibility with PostScript language Level 1 implementations. Also, a **limitcheck** error can occur because of insufficient memory for the requested imaging area.

Table 6.8 *Paper tray compatibility operators*

Operator	*PageSize*	*ImagingBBox*
lettertray †¶	[612 792]	null
legaltray¶	[612 1008]	null
ledgertray¶	[1224 792]	null
a3tray¶	[842 1191]	null
a4tray¶	[595 842]	null
b5tray¶	[516 729] or [499 709]	null
11x17tray¶	[792 1224]	null

The **lettertray** compatibility operator will be present only if the size [612 792] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **legaltray** compatibility operator will be present only if the size [612 1008] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **a4tray** compatibility operator will be present only if the size [595 842] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **b5tray** compatibility operator will be present only if the size [516 729] or the size [499 709] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

6.2.6 Page Duplex Compatibility Operators

All compatibility objects described below are defined in **statusdict** unless otherwise specified.

duplexmode – **duplexmode** *bool*

Returns the value of the page device parameter **Duplex**.

The page device parameter **Duplex** must be present for the compatibility operator **duplexmode** to be present.

Errors: **stackoverflow**

firstside – **firstside** *bool*

Returns *true* if the current page is a front side, *false* if the current page is a back side.

Note *This compatibility operator is sometimes found on products that do not support duplex printing. On these products, firstside may be used to generate output that is intended to be copied using a duplex copier.*

Errors: **stackoverflow**

newsheet – **newsheet** –

If **Duplex** is *true* and the current page is a back-side, causes this page to be printed as is (perhaps blank) and sets up a clean printing environment for the next page. Otherwise, executing **newsheet** has no effect.

The page device parameter **Duplex** must be present for the compatibility operator **newsheet** to be present.

Errors: None.

setduplexmode¶ *bool* **setduplexmode** –

This compatibility operator sets the page device parameter **Duplex** to *bool*.

The page device parameter **Duplex** must be present for the compatibility operator **setduplexmode** to be present.

Errors: **stackunderflow**, **typecheck**

settumble¶ *bool* **settumble** –

This compatibility operator sets the page device parameter **Tumble** to *bool*.

The page device parameter **Duplex** must be present for the compatibility operator **settumble** to be present.

Errors: **stackunderflow**, **typecheck**

tumble – **tumble** *bool*

Returns the value of the page device parameter **Tumble**.

The page device parameter **Duplex** must be present for the compatibility operator **tumble** to be present.

Errors: **stackoverflow**

6.2.7 Device Compatibility Operators

All device compatibility operators described below are defined in **systemdict**. The device operators aid in the management of any given file system.

devdismount †§ *string* **devdismount** –

This compatibility operator sets the device parameter **Mounted** to *false* within the parameter set corresponding to the device specified by *string*. It is necessary for the device to be mounted before it can be dismounted. Trying to dismount a device that is not mounted will have no effect. Some devices cannot be dismounted. Trying to dismount these will also have no effect.

In PostScript language Level 2, you can dismount from any save level if the **SystemParamsPassword** is not set. If it is set, **devdismount** will raise an **invalidaccess** error unless executed within an unencapsulated system administrator job.

Errors: **invalidaccess**, **stackunderflow**, **undefinedfilename**

devforall † *proc scratch* **devforall** –

devforall enumerates all known storage devices.

For each storage device, **devforall** copies its name into the supplied *scratch* string, pushes a string object that is the substring of *scratch* that was actually used, and calls *proc*. **devforall** does not return any results of its own, but *proc* may do so.

Note *Some of the storage devices enumerated by devforall correspond to communication channels. These will have a HasNames value equal to false.*

Errors: **invalidaccess**, **rangecheck**, **stackoverflow**, **stackunderflow**, **typecheck**, **undefined**

devformat †§ *string pages format* **devformat -**

This compatibility operator sets the **LogicalSize** device parameter of the parameter set corresponding to the device specified by string to the value specified by *pages*. It then sets the **InitializeAction**, in the same parameter set, to the value of *format*+1. Refer to the **InitializeAction** and **LogicalSize** file system device parameters for complete details.

Errors: **invalidaccess**, **limitcheck**, **rangecheck**, **stackunderflow**, **typecheck**, **undefined**, **undefinedfilename**

devmount †§ *string* **devmount** *bool*

This compatibility operator sets to *true* the **Mounted** device parameter boolean of the parameter set corresponding to the device specified by *string*. It then returns the resulting value of **Mounted** by reading it from the same parameter set. *True* indicates that the device was successfully mounted or was already mounted. *False* indicates that the device cannot be mounted at this time.

In PostScript language Level 2, you can mount from any save level if the **SystemParamsPassword** is not set. If it is set, **devmount** will raise an **invalidaccess** error unless executed within a unencapsulated system administrator job.

Errors: **invalidaccess**, **stackunderflow**, **undefinedfilename**

devstatus † *string* **devstatus** false (if device not found)

string **devstatus** *searchable writeable hasNames mounted removable searchOrder freePages size true* (if device found)

Takes a %disk% device name identified by *string* from the stack. If the device name is unknown, *false* will be left on the stack only. If the device name is found, it pushes various file system attributes for the device. The attributes are *searchable*, *writeable*, *hasNames*, *mounted*, *removable*, *searchOrder*, *freePages*, and *size*.

searchable The *searchable* attribute corresponds to the **Searchable** device parameter and is a boolean which indicates that the device will be searched when looking for a file with no device name prefix in its name.

writeable The *writeable* attribute corresponds to the **Writeable** device parameter and indicates whether files on this device can be written.

hasNames The *hasNames* attribute corresponds to the **HasNames** device parameter and is a boolean which indicates whether the device supports named files.

mounted The *mounted* boolean (**Mounted** device parameter) indicates whether the device is mounted.

removable The *removable* boolean (**Removable** device parameter) indicates whether the media within the device can be removed.

searchOrder The *searchOrder* attribute (**SearchOrder** device parameter) indicates the priority at which the device participates when searching for a file in operations in which no device has been specified.

freePages The *freePages* boolean (**Free** device parameter) indicates the amount of free space (in pages).

size The *size* attribute (**LogicalSize** device parameter) indicates the current size of the PostScript software file system (in pages).

Note *In Level 1, a "page" had a file system specific size (typically 1024).*

For a complete description of each of the device parameters mentioned above, refer to section 3.5.3.

Errors: **stackunderflow**

6.2.8 Imagesetter Compatibility Operators Found in Statusdict

accuratescreens ø – **accuratescreens** *boolean*

Returns the value of the user parameter **AccurateScreens**. A value of true means that accurate screening is enabled.

Errors: stackoverflow

checkscreen ø *freq angle* **checkscreen** *actualfreq actualangle moirelength*

Returns the actual screen frequency and angle that would be used if **setscreen** was called. The *moirelength* is the distance in inches where the deviation from the requested dot pattern would reach a fixed fraction of a cell size and is thus a measure of how accurate the actual screen would approximate the requested screen. Note that this operator does not affect the current screen.

Errors: stackoverflow

pagemargin ø – **pagemargin** *x*

Returns the *x* value (measured in device units) of the page device parameter **PageOffset**.

Errors: **stackoverflow**

pageparams ø – **pageparams** *width height margin orientation*

Suppose that the value of the page device parameter **PageSize** is [*x y*] and that **PageOffset** is [*X Y*]. Then **pageparams** returns values, depending on the value of the page device parameter **Orientation**, as indicated in the following table.

Orientation	width	height	margin	orientation
0	y	x	X	0
1	x	y	X	1
2*	y	x	X	0
3*	x	y	X	1

*Not applicable; not supported in Level 1 PostScript imagesetters.

Errors: **none**

setaccuratescreens ø *boolean* **setaccuratescreens** –

Sets the user parameter **AccurateScreens** to have the value *boolean*.

Errors: **stackunderflow**, **invalidaccess**, **typecheck**

setpage ø¶ *width height orientation* **setpage** –

Creates a new page device with the parameter **PageSize** set to [*width height*] and **Orientation** to orientation.

Errors: **limitcheck**, **rangecheck**, **stackunderflow**, **typecheck**

setpagemargin ø¶ *margin* **setpagemargin** –

Creates a new page device with the parameter **PageOffset** set to [*margin* 0].

Errors: **rangecheck**, **stackunderflow**, **typecheck**

setpageparams ø¶ *width height margin orientation* **setpageparams** –

Creates a new page device with the parameter **PageSize** set to [*width height*], the *x* value of **PageOffset** set to *margin*, and **Orientation** set to *orientation*.

Errors: **limitcheck**, **rangecheck**, **stackunderflow**, **typecheck**, **undefinedresult**

Appendix A

Changes Since Earlier Versions

A.1 Changes since Version 2016, July 7, 1995

- The new **CollateDetails** key has been added to the page device parameter set. See Table 2.1 on page 6.
- The new **LeadingEdge** key has been added to the page device parameter set. See Table 2.1 on page 6.
- The new **MediaClass** key has been added to the page device parameter set. See Table 2.1 on page 6.
- The key **OutputDevice** in Table 2.1 on page 6 has been modified. The reference to **OutputDeviceDict** has been removed.
- The **PageDeviceName** key description in the page device parameter set has been modified with a reference to the **MediaClass** key. See Table 2.1 on page 6.
- The new **RollFedMedia** key has been added to the page device parameter set. See Table 2.1 on page 6.
- A new table (Table 2.3 on page 19) describes some of the **CollateDetails** keys being used in products.
- A new table (Table 2.4 on page 20) describes some of the **PostRenderingEnhanceDetails** keys being used in products.
- A new section (Section 2.2) on **DeviceRenderingInfo** dictionaries has been added.
- Section 2.3.4, “The Fax Options Dictionary Keys,” has been clarified.
- The new key **CompressImageSource** has been added to the PostScript parameters. See section Table 3.2 on page 47.
- The new key **EnvironmentSave** has been added to the PostScript parameters. See Table 3.2 on page 47.

- The new key **InstalledRam** has been added to the PostScript parameters. See Table 3.2 on page 47.
- The key **MaxHWRenderingBuffer** has been modified to include ColorBurst. See Table 3.2 on page 47.
- The key **MaxPermanentVM** has been added to the PostScript system parameters. Table 3.2 on page 47.
- The key **PreferredServer** has been added to the set of parameters present in %PrintServer%. See Table 3.7 on page 79.
- The Table 3.17 “Parameters present in the %LAT% communications parameter set” on p. 100 has been added to section 3.5, “Device Parameters.”
- The key **TransmitEncapsulation** in Table 3.22 on page 107 has been modified.
- The key **HopCount** has been added to Table 3.24 on page 112.
- The key **TransmitEncapsulation** in Table 3.24 on page 112 has been modified.
- A RAM parameter table has been added to the “Disk, Cartridge, ROM and RAM Parameter Tables” on p. 117. The new table is Table 3.30 on page 124.
- The legal values of **Bus** in Table 3.28 on page 117 have been changed.
- An **ioerror** has been added for the **InitializeAction**, **Mounted**, and **PrepareAction** keys in %disk% and %cartridge% devices.
- The key **PrepareAction** has been added to Table 3.28 on page 117.
- A new section, section 3.5.6, “IDE Bus Parameter Set,” has been added.
- The keys **DarknessBlack**, **DarknessCyan**, **DarknessYellow** and **DarknessMagenta** have been added to Table 3.34 on page 130.
- The key **WaitTimeout** has been added to Table 3.38 on page 140.
- **GetPageDeviceName**, **GetHalfToneName** and **GetSubstituteCRD** in Table 4.1 on page 153 have been replaced with **ColorRendering**.
- Types 9 and 100 have been added to the enumeration of the **HalfToneType** implicit resource. (See Table 4.2 “Resources whose instances are implicit” on p. 155.)

- The description of **OutputDevice** in section 4.4, “Accessing Product Page Device Capability Information,” on page 157 has been modified.
- The instance **ColorBurst** has been added to Table 4.5 on page 159 and related text.
- Throughout Chapter 5, “Other Extensions to PostScript Language Level 2,” the type for **HalftoneName** has been changed from “string” to “name or string.”
- The key **Thresholds** in Table 5.4 on page 170 has been modified.
- A new section, section 5.5, “Synchronizing CRDs and ICC Profiles,” has been added on page 181.
- A paragraph describing the initial values of *D*, *E*, *F*, and *G* after execution of **setcolorspace** has been added to section 5.7, “Additional CIE-Based Color Spaces” on page 187.
- Section 5.8, “Fax Environment Interface,” has been moved from chapter 3, “Interpreter Parameters ” to chapter 5.

A.2 Changes since Version 2015, December 5, 1994

- The new **Jog** key has been added to the page device parameter set. See Table 2.1 on page 6.
- There is a better description of the **PageSize** entry of type 7 in the **Policies** dictionary. See Table 2.2 on page 17.
- The new key **DoPrintErrors** has been added to the PostScript system parameters. See Table 3.2 on page 47.
- There is a better description of %CommName_Pending%. See Communications Parameter Sets on page 61 and Predetermined Parameter Values on page 65.
- The **Filtering** key has been added to the set of parameters present in the %ScsiComm% communications parameter set. See Table 3.7 on page 79.
- Changes have been made to the **PrivateHost** and **TrapHost** keys in the set of parameters present in %SNMP%. See Table 3.18 on page 103.
- The new key **DialingPrefix** has been added to the parameters present in the %Fax%. See Table 3.41 on page 143.
- The new %FaxJobs% parameter set has been added. See Table 3.42 on page 149.

- The new **ReceiverCapabilities** key has been added to the fax job dictionary. See Table 5.10 on page 193.

- Four new keys (**ECMUsed**, **Format**, **Resolution** and **Speed**) have been added to the entries in a **Calls** dictionary.

- A diagram illustrating how to interpret the output returned by the **returnjoblist** operator has been added. See Figure 5.6 on page 193.

- Type 32 font dictionary has been added to the set of implicit **Font** resources. See section 5.3.1 on page 171 and Type 32 Font Dictionary on page 171.

- Three new operators have been added to the PostScript language for Type 32 fonts. They are **addglyph**, **removeall** and **removeglyph**. See New Type 32 Operators on page 173.

- The **HalftoneName** key has been added to the dictionary entries for all halftone types. See Changes Affecting All Halftone Types on page 165.

- The description of CID fonts has been improved. See CID Font Format on page 184.

- A description of the new CIE-based color spaces pre-extension has been added. See Additional CIE-Based Color Spaces on page 187.

- Two new compatibility operators have been added: **doprinterrors** and **setdoprinterrors**. See Compatibility Operators on page 207 and Using a Password to Change Persistent Values on page 210.

- The description of **processcolors** compatibility operator has been improved. See Using a Password to Change Persistent Values on page 210.

A.3 Changes since Version 2014, March 10, 1995

The following is a brief list of changes made to the *PostScript Language Reference Manual Supplement* since March 10, 1994. These changes were incorporated into the 2015 release.

- There is new support for CRD selection based on rendering intent. See the new page device key **PageDeviceName** found on page 13. See section 5.4 on page 176 for complete details on CRD selection. There are three new **ProcSets** associated with CRD selection. They are **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD**. See Table 4.1 on page 153.

- There is a better description of the **PageSize** matching rules found on page 13.

- The new **/Interpreter** key /PCL has been added to the list for LaserJet 4 emulation. See page 68. There is a new device parameter set called %PCL%. See page 137.

- The new key **PrinterControl** has been added to all device parameters sets of type /Communications. See page 70.

- The %Parallel% parameter set has new values for the **Handshake** key. There are also 2 new keys: **nAckPulseWidth** and **nStrobeExpectedPulseWidth**. See page 76.

- The %EtherTalk% parameter set has had the key **NodeID** added to it. See page 84.

- There is a new parameter set called %TokenTalk%. See page 85.

- Also for token ring, the new parameter set %TokenRingPhysical% has been added. See page 115.

- There is a new device parameter set called %Console%. See page 133. There is a new resource category called **Localization** which enumerates the possible natural languages that the console can be used with. See page 159.

- There are 2 new resource categories **PDL** and **ControlLanguage** which give information about language interpreted by the product. See page 160.

- In 2015, support for the new CID font format is present. See page 184 on page 184.

- Support for the new halftone type 10 has been added. See page 167

A.4 Changes since Version 2013, March 31, 1993

The following is a brief list of changes made to the *PostScript Language Reference Manual Supplement* since March 31, 1993. These changes were incorporated into the 2014 release.

- The default value has been added to the **HostJobID** key on page 29.

- The **PageCount** parameter on page 55 has an improved description.

- Section 3.5 (page 57) contains a new note to support the removal of the word “typically” from Tables 3.4 to 3.35.

- The description of **DelayedOutputClose** on page 66 explains better the parameter’s function.

- A description of the **Bus** key has been added to Table 3.7. See page 79.
- The **GatewayAddress** parameter has been updated to explain its reliance on the **IPAddressDynamic** parameter value. See page 108.
- The **Removable** parameter (page 121) is described in more detail.
- Descriptions of the new page device keys **DeferredMediaSelection**, **ImageShift**, and **MediaPosition** have been added to Table 2.1 on page 6. Also the **InsertSheet** and **PageOffset** descriptions are improved.
- 2.1, “Details Dictionaries” on page 18 describes better what happens if the **Type** key for a details dictionary is wrong or missing.
- The **DialCallee** parameter in the **FaxOptions** has the comma character better explained (page 28). Also in **FaxOptions**, there are 2 new keys. They are **HostJobID** (page 29) and **RecipientLanguage** (page 31). Also the description for **RecipientName** and **RecipientPhone** has been improved.
- 2.3.5, “CoverSheet, Confirmation and PageCaption Procedures,” has a new improved description.
- 2.5.3, “undefined Errors” explains that **undefined** is a possible error from **setpagedevice**.
- Two new imagesetter system parameters have been added to Table 3.2 on page 47. The new parameters **CurBufferType** (on page 47) and **MinBandBuffers** (page 55) are found in this table.
- The semantics of the **PageCount** system parameter have changed. See page 55.
- The new communication device parameter **DelayedOutputClose** is described on page 66. This parameter exists in almost all parameter sets of type /Communications.
- The description of the detection of protocol when using /AutoSelect has been clarified (page 69).
- The Novell SPX/IPX *Node address* syntax has been modified since the 2013 Supplement. Refer to Table 3.11 on page 88.
- The %AppSocket% parameter set has a new parameter called **ControlPortNumber** (page 92).

- Several new Novell device parameter sets are described. They are %RemotePrinter% (page 96), %PrintServer% (page 97), %SPX% (page 111), and %IPX% (page 112).

- The description for the **TrapHost** parameter in the %SNMP% set has changed. See page 103.

- The parameter set %UDP% (page 106) has been added to the family of TCP/IP related parameter sets.

- The description of the various parameters in the %IP% set has been rewritten. See Table 3.22 on page 107.

- The **Name** parameter in the %EthernetPhysical% set is read-only (page 114).

- The %Fax% parameter set has 2 new keys called **LocalLanguage** (page 145) and **Group3Adjustment** (page 144).

- 5.8.1, “Administrative Resources” has a new subsection called , “Job Queues which describes the new capability in 2014 of displaying fax job queues.

- 5.8.2, “Translations Dictionaries in the FaxDefaultProcs ProcSet” is new.

- Table 4.4, Description of keys present in an instance of the category OutputDevice on page 158 has had **ProcessColorModel** added to it.

- Table 4.5 on page 159 lists /WorldModem as a possible **Fax** instance value.

- There are several new instances listed in the **IODevice** resource category. The list includes **%RemotePrinter%**, **%PrintServer%**, **%UDP%**, **%SPX%**, and **%IPX%.** Each has the corresponding _NV and _Pending set names.

- There are other minor corrections throughout.

A.5 Changes since Version 2012, November 25, 1992

The following is a brief list of changes made to the *PostScript Language Reference Manual Supplement* since November 25, 1992. These changes were incorporated into the 2013 release.

- The new page device key **PageOffset** has been added to chapter 2.

- Section 3.5.2 has been reorganized to describe device parameter sets associated with both network and point-to-point communications. Several new parameter sets are described which have been defined to support the

use of the TCP/IP protocol over Ethernet. The new parameter sets include %LPR%, %AppSocket%, %Telnet%, %SNMP%, %SysLog%, %TCP%, %IP%, and %EthernetPhysical%.

- In 3.5.3, “File System Parameters” there is a subsection called, “Disk, Cartridge, ROM and RAM Parameter Tables” that now additionally describes the %rom% device parameter set.

- The LaserJet III emulator has several additional device parameters that had not previously been documented. Refer to Table 3.37 on page 137.

- The HP7475A emulator and the Diablo630 emulator device parameter sets both have a **Type** key which had previously not been documented. Refer to Table 3.39 and Table 3.40 on page 142. Also the Diablo630 has a **Pitch** key which had previously not been documented.

- Chapter 4 has been rewritten. Some resource instances had previously been missing. 4.5, “Accessing Product Hardware Options Information” describes the **HWOptions** resource instance found on some products. The **IODevice** resource category lists new instances for %LPR%, **%AppSocket%**, **%Telnet%**, **%SNMP%**, **%SysLog%**, **%TCP%**, **%IP%**, and **%EthernetPhysical%** which are some new TCP/IP and Ethernet related device parameter sets.

- 2013 PostScript products can optionally contain support for font Type 42. The Type 42 font format is a TrueType font with a PostScript wrapper to make it conform to the PostScript font model. TrueType is a font format originally developed by Apple Computer. Refer to section 5.3.2, “Type 42 Font Dictionary.”

- Various compatibility operators typically found on imagesetters and roll fed media devices have been added to chapter 6. The list includes **accuratescreens**, **checkscreen**, **mirrorprint**, **pagemargin**, **pageparams**, **resolution**, **setaccuratescreens**, **setpage**, **setpagemargin**, **setpageparams**, and **setresolution**.

A.6 Changes since Version 2011, January 24, 1992

The following is a brief list of changes made to the *PostScript Language Reference Manual Supplement* since January 24, 1992. These changes were incorporated into the 2012 release.

- A new chapter has been added, titled “Other Extensions to PostScript Level 2.” This chapter has a description of the new Type 6 halftone dictionary.

- Corrections have been made throughout the document. In addition, the text has been expanded in many places to provide more context for the reader.

- The new page device parameters introduced in 2012 are **DeviceRenderingInfo**, **FaxOptions**, **ProcessColorModel**, **SeparationColorNames**, and **SeparationOrder**.

- The Fax feature is described in detail in sections 2.2 and 3.6.

- Section 3.1 and 3.2 now give the reader better context when thinking about unencapsulated jobs and passwords.

- There is a new user parameter introduced in 2012 called **AccurateScreens**.

- There are several new system parameters introduced in 2012. They are **CurStoredFontCache**, **CurStoredScreenCache**, **MaxHWRenderingBuffer**, **MaxImageBuffer**, **MaxStoredFontCache**, and **MaxStoredScreenCache**.

- There are several new device parameter sets. The list includes %ScsiComm%, %os%, %Scsi%, %Engine%, %LaserJetIII%, %Fax%, and %Calendar%.

- The **Interpreter** key in all of the device parameter sets of type **/Communications** can now be set to **LaserJetIII** for PCL5 emulation. **EpsonFX850** is also a new choice.

- The **Interpreter** key in all of the device parameter sets of type /Communications can be set to /AutoSelect. When set to /AutoSelect, automatic and seamless switching between the available interpreters and emulators is enabled.

- In the %Serial% device parameter set, a new **FlowControl** choice is available called /XonXoff2.

- A new **Protocol** choice has been added for serial and parallel parameter sets called /TBCP.

- The new key **Filtering** has been added to the %LocalTalk% set.

- The new keys **Handshake** and **OutputDevice** have been added to the %Parallel% set.

- The device parameter sets of type **/FileSystem** have been changed to support removable media. A new key named **BlockSize** has been added. The %disk% parameter sets have the new keys **Bus** and **Interleave**.

- The new implicit resource category **HWOptions** has been added. There are some implicit **ProcSet**s for fax called **FaxOps** and **FaxAdminOps**. Also, **HalftoneType** can be set to 6.

- In the compatibility chapter, **processcolors** has been added.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z