PostScript® Language Reference Supplement

Adobe PostScript 3
Version 3010 and 3011
Product Supplement

Adobe® PostScript® 3™

30 August 1999

Adobe Systems Incorporated

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

Adobe Systems
Adobe House
West One Business Park
5 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

28-007

Contents

Figures

Tables

CHAPTER 1

Introduction

THIS DOCUMENT DESCRIBES extensions and features of the PostScript® language that are not documented in the *PostScript Language Reference*, Third Edition (referred to throughout this *Supplement* by the abbreviation *PLR3*). These include extensions to LanguageLevel 3 that are implemented in Adobe PostScript 3 versions 3010 and 3011, as well as some features that were too device-specific to include in the *PostScript Language Reference* and some facilities for maintaining backward compatibility with LanguageLevel 1 implementations.

- Chapter 2, “Page Device Parameters,” supplements Chapter 6 of the Third Edition. It describes additional page device parameters that are relatively device-specific—those that control features unique to a particular output device or limited to only a few devices.
- Chapter 3, “Interpreter Parameters,” supplements Appendix C of the Third Edition. It provides information about additional user, system, and device parameters.
- Chapter 4, “Resources,” supplements Section 3.9 of the Third Edition. It describes additional named resources that are relatively device-specific.
- Chapter 5, “Other Extensions,” provides information about miscellaneous extensions to the PostScript language that are not described in the Third Edition. These include new and extended font types, extensions to the contents of image dictionaries and trapping parameter dictionaries, and support for device color profiles as defined by the International Color Consortium (ICC).
- Chapter 6, “Compatibility,” supplements Chapter 8 of the Third Edition. It describes operators, procedures, and data values that are supported in LanguageLevel 2 and 3 implementations in order to maintain backward compatibility with LanguageLevel 1 implementations.

- Appendix A, "Changes since Earlier Versions," lists changes made to the PostScript language since version 3010. These changes were incorporated into the 3011 release.
- Appendix B, "Implementation Limits," supplements Appendix B of the Third Edition. It provides additional information about limits that are typical of PostScript interpreter implementations from Adobe Systems.
- Appendix C, "Emulator Parameter Sets," describes parameter sets of type Emulator, which provide supporting information for emulating non-PostScript imaging systems within a PostScript imaging system.
- Appendix D, "Typical Font Set," supplements Appendix E of the Third Edition. It lists the typical set of fonts available on most Adobe PostScript 3 products.
- Appendix E, "Updates to the PostScript Language Reference, Third Edition," describes content changes, additions, and corrections to the Third Edition.

The *Supplement* concludes with a Bibliography and an Index.

Note: *If you discover any errors in or have any problems with this documentation, please e-mail us at*

<http://doc_problems@adobe.com>

Please describe the error or problem as completely as possible and give us the document ID number (located at the foot of the cover page), the document title, and the page number or page range.

CHAPTER 2

Page Device Parameters

THIS CHAPTER SUPPLEMENTS Chapter 6 of the *PostScript Language Reference*, Third Edition. It provides information about additional page device parameters that are relatively device-specific—those that control features unique to a particular output device or limited to only a few devices.

2.1 Details Dictionaries

As discussed in *PLR3*, Section 6.1.2, some page device features are controlled by two separate parameters: a boolean or integer parameter that enables or disables the feature as a whole and a *details dictionary* containing variables that determine its precise behavior. For example, on a device with a stapling capability, the boolean page device parameter **Staple** specifies whether to staple a document after printing, while the details dictionary **StapleDetails** might contain further information such as the number, location, and orientation of the staples.

The exact contents of the details dictionary for a given feature may vary from one device to another, depending on the specific capabilities and characteristics of the device. If no further configuration is possible beyond simply enabling or disabling the feature, only the boolean parameter will be present with no associated details dictionary. Applications should never assume the presence of a details dictionary without knowing the exact nature of the output device with which they are communicating.

Every details dictionary has a **Type** entry whose integer value determines the remaining structure and contents of the dictionary. The **setpagedevice** operator examines this entry and raises an **undefined** error if it is absent or a configuration error if it doesn't match the value expected by the current page device. For a given feature, details dictionaries with the same **Type** value will have entries with exactly the same names, formats, and meanings, even if intended for use with different output devices. When the values in a details dictionary are changed, the **Type** entry also determines whether the contents of the new dictionary overwrite those of the current one or are merged with it. To avoid conflicts, details dictionaries and their associated **Type** values should be registered with Adobe.

2.2 Parameter Descriptions

Table 2.1 lists the page device parameters covered in this section. Note, however, that not all parameters described here are applicable to all devices; consult the product documentation for a particular device to see exactly which parameters it supports. In the future, new parameters may be defined as needed to control new processing options or device features. Once defined for any device, a given parameter name will always be used for the same feature on any subsequent devices that support it.

TABLE 2.1 Categories of additional page device parameters

CATEGORY	DESCRIPTION	PARAMETERS	
Media selection	Select the appropriate type of paper or other physical medium.	**ManualFeedTimeout**	
Page image placement	Specify how page images are to be rendered onto the physical medium.	**Signature** **SignatureDetails**	
Rendering control	Specify device-specific settings to control the rendering of page images onto the physical medium.	**DeviceRenderingInfo** **PostRenderingEnhance** **PostRenderingEnhanceDetails** **PreRenderingEnhance** **PreRenderingEnhanceDetails**	
Page delivery	Control the disposition of physical output.	**Bind** **BindDetails** **Booklet** **BookletDetails** **CollateDetails** **Fold** **FoldDetails** **Laminate**	**OutputPage** **Punch** **PunchDetails** **SlipSheet** **Staple** **StapleDetails** **Trim**

TABLE 2.1 Categories of additional page device parameters

CATEGORY	DESCRIPTION	PARAMETERS
Error handling	Specify the device's response to various error conditions.	**ExitJamRecovery**

2.2.1 Media Selection

Table 2.2 describes an additional page device parameter related to media selection, supplementing those described in Section 6.2.1 of the *PostScript Language Reference*, Third Edition.

TABLE 2.2 Additional page device parameter related to media selection

KEY	TYPE	VALUE
ManualFeedTimeout	integer	The number of seconds to wait for a page to be fed manually before generating a **timeout** error. A value of 0 means no timeout (infinite wait).

2.2.2 Page Image Placement

Table 2.3 describes additional page device parameters related to page image placement, supplementing those described in Section 6.2.3 of the *PostScript Language Reference*, Third Edition.

TABLE 2.3 Additional page device parameters related to page image placement

KEY	TYPE	VALUE
Signature	boolean	A flag determining whether page images are to be placed on the output medium in the form of *signatures* (sets of pages arranged on the same sheet of physical medium in such a way that they will appear in the correct order and orientation when the medium is folded and cut). How this process is performed is device-dependent. On some devices, the device itself provides the memory and disk space needed to collect a set of page images and arrange them into a signature; on other devices, the interpreter may have to rearrange the pages and deliver them to the device as a single image representing the entire signature. In the latter case, the interpreter may need to execute multiple page descriptions and store the results in order to combine them into a

		signature image. This use of **Signature** is supported by relatively few devices and is subject to resource limits in those that do support it.
SignatureDetails	dictionary	A dictionary containing device-specific parameters that determine how the page images are to be arranged in signatures. This parameter is ignored if the value of **Signature** is *false*.

2.2.3 Rendering Control

Some output devices need the ability to specify additional, device-specific settings to control the rendering of page images onto the physical output medium. The nature and meaning of these settings may be very different for different devices. The page device parameters described in Table 2.4 allow the specification of such device-specific rendering information.

TABLE 2.4 Additional page device parameters related to rendering control

KEY	TYPE	VALUE
DeviceRenderingInfo	dictionary	A dictionary containing device-specific rendering parameters; see discussion below and Table 2.5.
PreRenderingEnhance	boolean	A flag that enables or disables image enhancement before the page image is rasterized in memory.
PreRenderingEnhanceDetails	dictionary	A dictionary containing device-specific parameters that control the operation of prerendering image enhancement on this device. This parameter is ignored if the value of **PreRendering-Enhance** is *false*.
PostRenderingEnhance	boolean	A flag that enables or disables image enhancement after the page image is rasterized in memory.
PostRenderingEnhanceDetails	dictionary	A dictionary containing device-specific parameters that control the operation of postrendering image enhancement on this device. This parameter is ignored if the value of **PostRendering-Enhance** is *false*.

DeviceRenderingInfo is an optional dictionary holding device-specific information to control the rendering process. Like a details dictionary (see Section 2.1, “Details Dictionaries”), the **DeviceRenderingInfo** dictionary has a **Type** entry whose integer value determines its structure and contents. If two different devices use **DeviceRenderingInfo** dictionaries with the same **Type**, then the dictionaries will have exactly the same named entries and the form and meaning of each entry will be the same. This allows an application to manipulate the contents of the **DeviceRenderingInfo** dictionary based solely on the value of its **Type** entry, without needing to know anything further about the exact nature of the output device. Note, however, that **DeviceRenderingInfo** is *not* a details dictionary and is not enabled or disabled by a boolean device parameter; its contents are simply applied as needed to control the rendering process on the device.

Besides the **Type** entry, Table 2.5 shows two other typical **DeviceRenderingInfo** entries commonly used by PostScript devices. These are only examples; a given output device may include other entries in its **DeviceRenderingInfo** dictionary, and may or may not include the ones described here. See product-specific documentation for a complete description of the dictionary's contents for any particular device.

TABLE 2.5 Typical entries in a DeviceRenderingInfo dictionary

KEY	TYPE	VALUE
Type	integer	*(Required)* A code that determines the format and meaning of the other entries in the dictionary.
ValuesPerColorComponent	integer	The number of possible values for each color component, or the number of gray levels on a monochrome device.
Limit	integer	The maximum colorant density to be applied to the output medium. The value is expressed as a percentage in the range 100–400, where 400 represents the maximum possible density on a device with four process colorants.

In addition to using **DeviceRenderingInfo** to control the rendering process itself, some devices may wish to perform special processing either before rendering (to set up conditions for the rendering process) or after it (to do image enhancement on the results). The page device parameters **PreRenderingEnhance** and **PostRenderingEnhance** enable or disable such processing; the corresponding details dictionaries **PreRenderingEnhanceDetails** and **PostRenderingEnhanceDetails** can hold additional, device-specific parameters to control it. As usual, these dictionaries must have a **Type** entry to characterize their structure and contents. The remaining entries are device-dependent, and are described in product-specific documentation; Table 2.6 shows some typical entries included in some devices' **PostRenderingEnhanceDetails** dictionaries.

TABLE 2.6 Typical entries in a PostRenderingEnhanceDetails dictionary

KEY	TYPE	VALUE
Type	integer	*(Required)* A code that determines the format and meaning of the other entries in the dictionary.
PrintQuality	integer	An integer specifying the level of output quality desired. A zero value denotes the lowest available level, or "draft" quality.
TonerSaver	integer	An integer specifying the degree to which colors should be lightened to economize on the use of colorants. A zero value denotes no lightening; nonzero values reduce the tint levels of colorants applied to the output medium, causing (for example) full black to appear gray.
REValue	integer	An integer specifying the level of rendering enhancement desired. A zero value denotes the lowest available level of enhancement, typically none at all.

2.2.4 Page Delivery

Table 2.7 describes additional page device parameters related to page delivery, supplementing those described in Section 6.2.4 of the *PostScript Language Reference*, Third Edition.

TABLE 2.7 Additional page device parameters related to page delivery

KEY	TYPE	VALUE
OutputPage	boolean	A flag specifying whether to produce actual physical output. If the value of this parameter is *false*, all processing, including rasterization to a frame buffer, is performed as if the page were to be printed, but no physical output is produced. The time required to process a page thus includes everything except the time spent waiting for the marking engine; this is useful for measuring the cost of page execution.
CollateDetails	dictionary	A dictionary containing device-specific parameters that determine how the output is to be collated. See Table 2.8 for examples of typical contents. This parameter is ignored if the value of **Collate** (*PLR3*, pp. 417–418) is *false*.
Fold	integer	A code specifying whether and when to fold the output: 0 Do not fold. 1 Fold at device deactivation. 2 Fold at the end of the job.

		3 Fold after each page set, as defined by the **Collate** parameter (*PLR3,* p. 419). 4 Fold after each **showpage** or **copypage** operation.
FoldDetails	dictionary	A dictionary containing device-specific parameters that determine how the output is to be folded. This parameter is ignored if the value of **Fold** is *false.*
Trim	integer	A code specifying whether and when to trim the output: 0 Do not trim. 1 Trim at device deactivation. 2 Trim at the end of the job. 3 Trim after each page set, as defined by the **Collate** parameter (*PLR3,* p. 419). 4 Trim after each **showpage** or **copypage** operation.
Staple	integer	A code specifying whether and when to staple the output: 0 Do not staple. 1 Staple at device deactivation. 2 Staple at the end of the job. 3 Staple after each page set, as defined by the **Collate** parameter (*PLR3,* p. 419). 4 Staple after each **showpage** or **copypage** operation.
StapleDetails	dictionary	A dictionary containing device-specific parameters that determine how the output is to be stapled. This parameter is ignored if the value of **Staple** is *false.*
Booklet	boolean	A flag specifying whether to fold, trim, and staple the output into booklet form.
BookletDetails	dictionary	A dictionary containing device-specific parameters that determine how the output is to be folded, trimmed, and stapled into booklet form. This parameter is ignored if the value of **Booklet** is *false.*
Bind	integer	A code specifying whether and when to bind the output: 0 Do not bind. 1 Bind at device deactivation. 2 Bind at the end of the job.

		3 Bind after each page set, as defined by the **Collate** parameter (*PLR3,* p. 419). 4 Bind after each **showpage** or **copypage** operation.
BindDetails	dictionary	A dictionary containing device-specific parameters that determine how the output is to be bound. This parameter is ignored if the value of **Bind** is *false*.
Punch	integer	A code specifying whether and when to punch the output: 0 Do not punch. 1 Punch at device deactivation. 2 Punch at the end of the job. 3 Punch after each page set, as defined by the **Collate** parameter (*PLR3,* p. 419). 4 Punch after each **showpage** or **copypage** operation.
PunchDetails	dictionary	A dictionary containing device-specific parameters that determine how the output is to be punched. This parameter is ignored if the value of **Punch** is *false*.
Laminate	boolean	A flag specifying whether to laminate the output page. How a page is laminated is device-dependent.
SlipSheet	integer	A code specifying whether and when to insert a slip sheet into the output: 0 Do not insert slip sheets. 1 Insert a slip sheet at device deactivation. 2 Insert a slip sheet at the end of the job. 3 Insert a slip sheet after each page set, as defined by the **Collate** parameter (*PLR3,* p. 419). 4 Insert a slip sheet after each **showpage** or **copypage** operation. A slip sheet may be, for example, a sheet of colored paper whose purpose is to separate multiple copies visually; the selection of specific media is device-dependent. No visible marks can be rendered on a slip sheet; its imageable area is set to an empty region. A slip sheet may pass through the device's usual imaging mechanism (such as the fuser assembly on a laser printer) or may be routed via a separate paper path. Compare the **InsertSheet** parameter (*PLR3*, p. 402.)

Table 2.8 shows, in addition to the required **Type** entry, another typical **Collate-Details** entry commonly used by PostScript devices. This is only an example; a

given output device may include other entries in its **CollateDetails** dictionary, and may or may not include the one described here. See product-specific documentation for a complete description of the dictionary's contents for any particular device, as well as those of other details dictionaries (**FoldDetails**, **BookletDetails**, **BindDetails**, **StapleDetails**, **PunchDetails**). Note also that additional details dictionaries not currently supported, such as **TrimDetails** or **LaminateDetails**, could be included if needed by a particular product.

TABLE 2.8 Typical entries in a CollateDetails dictionary

KEY	TYPE	VALUE
Type	integer	*(Required)* A code that determines the format and meaning of the other entries in the dictionary.
ProofSet	boolean	A flag specifying whether to generate a proof copy of the output job. If the value of this parameter is *true*, the device will print an initial copy of the collated job for the user's inspection and will then remain in a "hold" state until some device-specific action is taken, such as confirming or aborting the printing of the remaining copies.

2.2.5 Error Handling

Table 2.9 describes an additional page device parameter related to error handling, supplementing that described in Section 6.2.7 of the *PostScript Language Reference*, Third Edition.

TABLE 2.9 Additional page device parameter related to error handling

KEY	TYPE	VALUE
ExitJamRecovery	boolean	A flag specifying whether pages that jam in the exit path should be reprinted. Disabling jam recovery may improve performance by allowing a greater degree of overlap in page processing.

2.3 Envelope Orientation in User Space

This section describes how default user space is oriented relative to the flap on an envelope. The following discussion assumes that the device's **Install** procedure (*PLR3*, Section 6.2.6) does not alter the default transformation matrix.

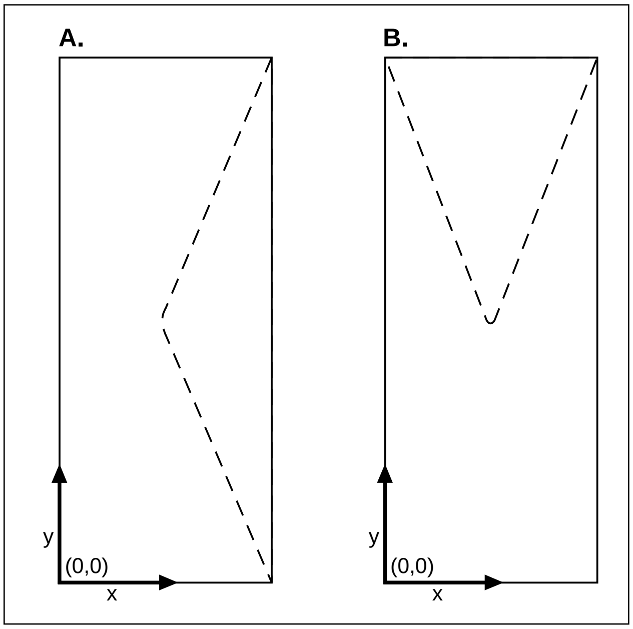

FIGURE 2.1 *Default user space for an envelope in portrait orientation*

As described in *PLR3*, p. 401, the page device parameter **PageSize** holds an array of two numbers, [*width height*], specifying the overall dimensions of the physical output medium. If the medium is in portrait orientation (*width* < *height*), the default user space is set up as shown in Figure 2.1. The dashed lines in the figure indicate that the flap of the envelope is on the side facing down. In Figure 2.1A, the flap lies along the long edge of the envelope; in Figure 2.1B it is along the short edge. In both cases, the default user space is set up with its origin on the edge of the envelope opposite the flap and in the corner diagonally opposite the return address (on a U. S. business envelope). If the **PageSize** parameter specifies landscape orientation (*width* > *height*), the default user space is rotated 90 degrees counterclockwise from the portrait orientation shown in the figure.

CHAPTER 3

Interpreter Parameters

THIS CHAPTER SUPPLEMENTS Appendix C of the *PostScript Language Reference*, Third Edition. It provides information about additional user, system, and device parameters.

Much of this chapter concerns communication between an imaging system and a host. These terms are defined as follows:

- An *imaging system* is a computer system containing a PostScript interpreter that is in direct control of a print engine or other raster output device, and that receives PostScript jobs to execute via one or more communication channels. The communication channels might be, for example, serial or parallel ports, or (for network communications) Ethernet connections.
- A *host* is a computer system, such as a personal computer or workstation, connected to an imaging system through one or more communication channels. The host—via an application, or perhaps more specifically a driver or spooler—sends PostScript jobs over the communication channel to the imaging system, which executes the jobs.
- The term *unit* is used in this chapter to refer more broadly to any device connected to a network (an imaging system or some other type of device).

3.1 User Parameters

The following user parameters are described in *PLR3*, Table C.1:

AccurateScreens	**MaxFontItem**	**MaxSuperscreen**
HalftoneMode	**MaxFormItem**	**MaxUPathItem**
IdiomRecognition	**MaxLocalVM**	**MinFontCompress**
JobName	**MaxOpStack**	**VMReclaim**
MaxDictStack	**MaxPatternItem**	**VMThreshold**
MaxExecStack	**MaxScreenItem**	

Table 3.1 describes two additional user parameters: **JobTimeout** and **Wait-Timeout**.

TABLE 3.1 User parameters

KEY	TYPE	VALUE
JobTimeout	integer	The *job timeout*—that is, the number of seconds a job is allowed to execute before it will be aborted and a PostScript **timeout** error generated. The value of **JobTimeout** is decremented during the job; reading it returns the number of seconds remaining before the job timeout will occur. Time spent waiting for communications and correcting device error conditions is not considered part of the job execution time. The minimum value for **JobTimeout** is 15; supplying a number in the range 1 to 14 will set the parameter to 15. (Allowing smaller values could prevent a subsequent job from successfully setting **JobTimeout** to another value.) A negative value will be ignored and the previous setting of **JobTimeout** used. Setting this parameter to 0 disables job timeout altogether (essentially signifying that the timeout is infinite). Alterations to **JobTimeout** are not subject to **save** and **restore**. It is initialized to the value of the **JobTimeout** system parameter at the beginning of each job.
WaitTimeout	integer	The *wait timeout*—that is, the number of seconds the interpreter waits to receive additional data from the host before it aborts the current job by executing a PostScript **timeout** error. A negative value will be ignored and the previous setting of **JobTimeout** used. Setting this parameter to 0 disables wait timeout altogether (essentially signifying that the timeout is infinite).

3.2 System Parameters

The following system parameters are described in *PLR3*, Table C.2:

BuildTime	**FontResourceDir**	**MaxSourceList**
ByteOrder	**GenericResourceDir**	**MaxStoredScreenCache**
CurDisplayList	**GenericResourcePathSep**	**MaxUPathCache**
CurFontCache	**LicenseID**	**PageCount**
CurFormCache	**MaxDisplayAndSourceList**	**PrinterName**
CurOutlineCache	**MaxDisplayList**	**RealFormat**
CurPatternCache	**MaxFontCache**	**Revision**
CurScreenStorage	**MaxFormCache**	**StartJobPassword**
CurSourceList	**MaxImageBuffer**	**StartupMode**
CurStoredScreenCache	**MaxOutlineCache**	**SystemParamsPassword**
CurUPathCache	**MaxPatternCache**	
FactoryDefaults	**MaxScreenStorage**	

Table 3.2 describes additional system parameters.

TABLE 3.2 System parameters

KEY	TYPE	VALUE
CompressImageSource	boolean	A flag specifying whether a compression filter will be applied to the image source data where it would benefit memory usage or performance.
CurBufferType	name	*(Read-only)* A name object specifying how raster memory is used: Band — The system will render to bands (groups of scan lines), regardless of the amount of RAM available. Hybrid — If the **MaxRasterMemory** system parameter 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. This parameter is typically found on imagesetters.
CurInputDevice	string	*(Read-only)* The name of the communications device corresponding to the current input file for the currently executing PostScript job. This value corresponds to the Communication parameter set whose values are normally stored in RAM—for example, %Serial%. See Section 3.3.3, "Communications Parameter Sets."

CurOutputDevice	string	*(Read-only)* Similar to **CurInputDevice**, but corresponds to the current output file.
DoPrintErrors	boolean	A flag specifying whether the built-in error handler for the product is enabled. All PostScript imaging systems have an error handler to catch errors that are generated by programs (see *PLR3*, Section 3.11.2). Some imaging systems also have a built-in error handler that can be enabled to print information about the error; **DoPrintErrors** is present only in imaging systems that have such an error handler. Any imaging system that supports a LaserJet 4 or later emulation will have one, which can be controlled not only with this parameter but also with the PJL command `@PJL [SET | DEFAULT] LPARM: POSTSCRIPT PRTPSERRS = [ON | OFF]` The **$error** dictionary will contain information about the error that an implementation might want to report in its printed error message, such as the contents of the operand, execution, and dictionary stacks at the time of the error. For an example, see Appendix A of *PostScript Language Program Design*.
DoStartPage	boolean	A flag specifying whether the start page should be printed during system startup. Some printers print a start page when the power is turned on to identify the printer and how it is configured.
EnvironmentSave	boolean	A flag that applies to systems with multiple page description language (PDL) interpreters and specifies whether permanent objects belonging to all the interpreters persist across PDL switches. If this flag is *false*, the permanent objects do not persist, and all memory belonging to dormant PDL interpreters can potentially be reclaimed when the system runs out of memory. In low-memory configurations, this parameter should always be *false*, since setting it to *true* could make the system essentially unusable. See also **MaxPermanentVM**. When the value of **EnvironmentSave** is changed, the new value is effective immediately (the system does not have to be restarted).
FatalErrorAddress	integer	The address at which a fatal system software error occurred, or 0 if no such error occurred. A fatal error causes the 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 occurred is stored in this parameter and is transmitted to the host over the communication channel. Thus, a nonzero value of **FatalErrorAddress** indicates that a fatal error occurred earlier. On some products, if this value is nonzero during system initialization, the address is printed on the start page or possibly on a separate page.

InstalledRam	integer	*(Read-only)* The number of bytes of installed RAM in the product. **InstalledRam** should not be confused with **RamSize**, which is the amount of RAM available to the page description language.
JobTimeout	integer	The value to which the user parameter **JobTimeout** is initialized at the beginning of each job. See **JobTimeout** in Table 3.1 on page 14.
MaxHWRenderingBuffer	integer	The number of bytes to reserve for use by hardware rendering devices, such as PixelBurst or ColorBurst, to store display lists. The memory is permanently allocated during system initialization. Supplying a value that is outside the valid range (which is product-dependent) sets the parameter to the nearest allowable value. The minimum value for this parameter (typically 8192) meets the requirements of the rendering device, and the maximum value is an amount that will not jeopardize execution of a PostScript job.
MaxPermanentVM	integer	The maximum number of bytes of permanent VM that can be downloaded when **EnvironmentSave** is *true*. This is the amount of VM at **save** level 0 defined by unencapsulated PostScript jobs. Any attempt to download more permanent VM than this maximum will generate a **VMerror**. Supplying a value that is less than the smallest allowable value, or less than the current amount of permanent VM (plus a small threshold value), sets the maximum to the smallest allowable value. **MaxPermanentVM** is present in the system parameter set only if **EnvironmentSave** is defined.
MaxRasterMemory	integer	The maximum number of bytes occupied by raster memory. Unused raster memory is available for use as virtual memory; thus, this parameter makes it possible to trade off raster memory (which allows larger page sizes and higher resolutions) for virtual memory (which allows more downloaded fonts and more complex pages). **MaxRasterMemory** is consulted only during system initialization; any changes to the value of this parameter will not take effect until then.
MinBandBuffers	integer	The minimum number of band buffers, each the size of a rendered band (group 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. This parameter is typically found on imagesetters, where the default is 2 for configurations with up to 32 MB of memory, or 3 otherwise.
RamSize	integer	*(Read-only)* The number of bytes of installed RAM available to the page description language. In some cases, this value might be less than the total amount of installed RAM in the product (**InstalledRam**). For exam-

		ple, the system diagnostics might have determined that certain banks of RAM are defective, so it would consider them unavailable.
SourceListBypass	boolean	A flag that affects the one-operand (dictionary) form of the **image** and **imagemask** operators. If this flag is *true*, **image** and **imagemask** may delay reading data from the source identified by a **DataSource** dictionary entry in certain product-specific cases; otherwise, these operators will immediately read the data from the data source. The data source content will determine what image is painted and whether errors are generated; the existence or value of **SourceListBypass** will have no effect on what is painted, but it can improve performance if set to *true*.
ValidNV	boolean	*(Read-only)* A flag specifying whether nonvolatile memory is being used to store values of persistent parameters. The implementation is product-specific. A common approach is that if nonvolatile memory is corrupt during system initialization, the parameters are initialized with factory default values. Subsequent testing is done to confirm whether nonvolatile memory is still corrupt; if it is, **ValidNV** is set to *false*, and parameter values will not be read from and written to nonvolatile memory thereafter. In many products, if nonvolatile memory is corrupt it is emulated in RAM. The operating behavior is the same, except that persistent parameter values are lost when the imaging system is powered off or restarted (in which case factory defaults are used).
WaitTimeout	integer	The value to which the user parameter **WaitTimeout** is initialized at the beginning of each job. See **WaitTimeout** in Table 3.1 on page 14.

3.3 Device Parameters

This section augments the documentation of device parameters provided in Section C.4 of the *PostScript Language Reference*, Third Edition.

Note: *Not all of the device parameters described in this* Supplement *will be present in every product.*

3.3.1 Device Parameter Sets

Device parameters are grouped into named *parameter sets* (dictionaries) that describe the configuration of a physical or logical communication channel, storage device, hardware device, or software entity (such as a language emulator). For example, the parameter set named %Serial% contains the parameters for a serial device. Each parameter set known to the **setdevparams** and **currentdevparams** operators corresponds to an instance of the **IODevice** resource category. Even if two products have devices with the same name, their respective parameter sets might differ because the hardware support for that device may differ on the two products.

Every device parameter set must include a read-only parameter that indicates its type. The key for this parameter is **Type**, and its value is one of the following four values:

- Communications or Parameters. Adobe has defined various device parameter sets of these two types to aid system administrators in setting up and maintaining network imaging systems. These parameter sets are based on layers within the OSI (Open Systems Interconnection) model, an international standard for how messages should be transmitted between any two points in a telecommunications network, as shown in Figure 3.1. Communications parameter sets are associated with the application layer of OSI or with point-to-point connections that need not be layered or conform to OSI. Parameters parameter sets correspond to the transport, network, data link, and physical layers of OSI, as well as to miscellaneous devices such as the print engine and the calendar.
- FileSystem. Parameter sets of this type correspond to storage devices.
- Emulator. Parameter sets of this type allow PostScript imaging systems to emulate other imaging systems.

Communications, Parameters, and FileSystem parameter sets are described in detail in this chapter. Emulator parameter sets are described in Appendix C.

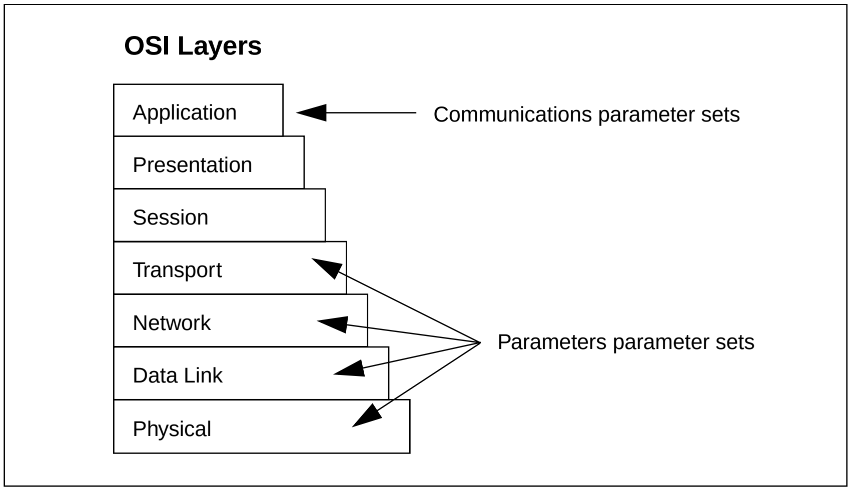

FIGURE 3.1 *OSI (Open Systems Interconnection) layers*

3.3.2 Device Parameter Interdependencies

Device parameters differ from user and system parameters in that they are interdependent; that is, the acceptability of a value for a given parameter may depend on the value of another parameter. For example, the serial communications device parameter set contains the parameters **Interpreter** and **Protocol: Interpreter** determines which page description language is to be used for an incoming job on that channel, and **Protocol** determines the communications protocol used to send and receive data. Although PostScript is a valid value for **Interpreter** and Raw is a valid value for **Protocol**, the serial channel cannot be configured to have both of these values set at the same time; a **configurationerror** would occur.

Most interdependencies are among parameters in the same device parameter set. However, there is a dependency among all communications devices that requires both the **On** and **Enabled** flags to be *true* in at least one of the communication channels. There might also be cases of interdependencies between certain device parameter sets. For example, if LocalTalk and a serial channel share the same hardware port on an imaging system, both cannot have their **On** flag set to *true* at the same time. If this flag is set to *true* in one of these channels when it is already *true* in the other channel, it is set to *false* in the latter channel.

3.3.3 Communications Parameter Sets

Parameter sets of type Communications are associated with the application layer of OSI, from which the PostScript interpreter or emulator receives its jobs, or with point-to-point connections that need not be layered or conform to OSI. (The OSI model is illustrated in Figure 3.1.)

A raster output device can have various physical communication channels and can use many different protocols over these channels. The hardware choices include:

- For point-to-point connections, serial, parallel (unidirectional or bidirectional), and SCSI bus
- For network communications, LocalTalk, Ethernet, and TokenRing

In addition, network adapters, such as line multiplexers, parallel-to-Ethernet adapters, and SCSI-to-Ethernet adapters, allow raster output devices to connect to networks to which they do not have a physical connection.

For point-to-point communications, there are no underlying device parameters specified by Parameters parameter sets (that is, associated with the lower levels of the OSI model): only the Communications parameter set is required. For network communications, however, a combination of Communications and Parameters parameter sets is required to fully specify all layers of the OSI model. The network communications parameter sets apply to raster output devices that are true peers of the network and have a direct connection to it; the Parameters parameter sets support the underlying communications protocols.

To reside directly on networks, raster output devices must recognize various standard communications protocols, including Novell NetWare (SPX/IPX), AppleTalk, and TCP/IP. The choices of protocols and the physical media on which they reside are limited, as shown in Figure 3.2. The connectors in this diagram indicate that the network protocols can be used with the physical medium indicated to deliver and receive messages.

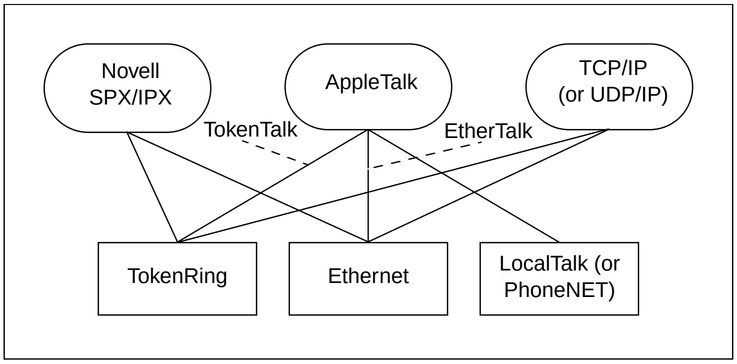

FIGURE 3.2 *Relationship between network communications protocols and physical communications media*

The protocols illustrated in Figure 3.2 are described in further detail below. For more information on the books referred to here, see the Bibliography.

- *Novell SPX/IPX*. The Novell NetWare communications protocols were originally derived from the XNS (Xerox Network System) protocols. For example, the Novell IPX (Internetwork Packet Exchange) protocol is virtually identical to the Xerox Network Layer Protocol called IDP (Internetwork Datagram Protocol). For a discussion of Novell networks, see *Analyzing Novell Networks*.
- *AppleTalk*. AppleTalk is the name Apple® Computer uses for its networking software that is built into every Macintosh® computer. The data link protocol LocalTalk Link Access Protocol (LLAP) is used for communicating over LocalTalk or PhoneNET networks. EtherTalk Link Access Protocol (ELAP) and TokenTalk Link Access Protocol (TLAP) are the data link protocols for communication over Ethernet and TokenRing, respectively. AppleTalk over Ethernet is called EtherTalk, and AppleTalk over TokenRing is called TokenTalk. Datagram Delivery Protocol (DDP), a layer on top of the data link protocol, provides delivery of AppleTalk packets across networks. See *Computer Networks* for a description of TokenRing and Ethernet, and *Inside AppleTalk* for a description of AppleTalk.
- *TCP/IP (or UDP/IP).* TCP/IP is a network architecture that is sponsored by DARPA (the Defense Advanced Research Projects Agency) and has been adopted by a large number of vendors. TCP (Transmission Control Protocol) is a transport layer protocol, and IP (Internet Protocol) 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.

Setting Up Communications Parameter Sets

The communications parameters for a product can be set up in different ways and may involve the use of front panels, hardware switches, and PostScript operators and procedures. This section describes a generic model for setting communications parameters that works across a variety of products and enables PostScript spoolers and utilities to use the same model when reading and writing communications device parameters.

A raster output device typically has several hardware ports for communications. There can be multiple physical ports of the same type, in which case each has its own parameter set. When there are multiple instances of a particular Communications parameter set, the naming convention for instances beyond the first is to add the letter B for the second instance, C for the third, and so on. If there were two serial ports, for example, their parameter sets would be named %Serial% and %SerialB%.

Furthermore, multiple parameter sets can be associated with the same physical port if that port can serve different roles, such as the serial port named channel B in an imaging system that has the following three ports:

- A parallel port, associated with the parameter set named %Parallel%.
- A serial port named channel A, wired to a 25-pin RS-232A connector and associated with the parameter set named %Serial%.
- A port named channel B, wired to either an 8-pin or a 9-pin connector and associated with two parameter sets named %SerialB% and %LocalTalk%. This port can serve as either a serial port or a LocalTalk port, depending on which of the two parameter sets has its **On** parameter set to *true*.

%CommName%, %CommName_NV%, and %CommName_Pending%

To provide a consistent model that is product-independent, all communications devices have three parameter sets. If the name of the device is, for example, %Serial%, the three associated parameter sets are named %Serial%, %Serial_NV%, and %Serial_Pending%. These names are shown in general here as

%*CommName*%, %*CommName*_NV%, and %*CommName*_Pending% (where *CommName* is the part of the name that varies by device). These three parameter sets have the following general characteristics:

- %*CommName*% values are usually stored in RAM and do not persist when the imaging system is powered off.
- %*CommName*_NV% values are usually stored in nonvolatile memory. If a product has limited nonvolatile memory, only some of the %*CommName*_NV% values may actually be saved to nonvolatile memory; otherwise, typically all writeable %*CommName*_NV% values are saved to nonvolatile memory. If the intent is to affect persistent values, PostScript utility programs should use %*CommName*_NV%, and the implementation will do the best it can given the amount of nonvolatile memory available in the product.
- %*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 received from a communication channel. This parameter set reflects either the current values of a writeable 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 in the next section.

In our imaging system example above, the following parameter sets might be available:

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

Note: *On some products, the parameter sets* %CommName%, %CommName_*NV%, and* %CommName_*Pending% may not be distinct from one other.*

Hierarchy of Communications Parameter Sets

Figure 3.3 illustrates the hierarchical relationship that exists among the three parameter sets %*CommName*%, %*CommName*_NV%, and %*CommName*_Pending%.

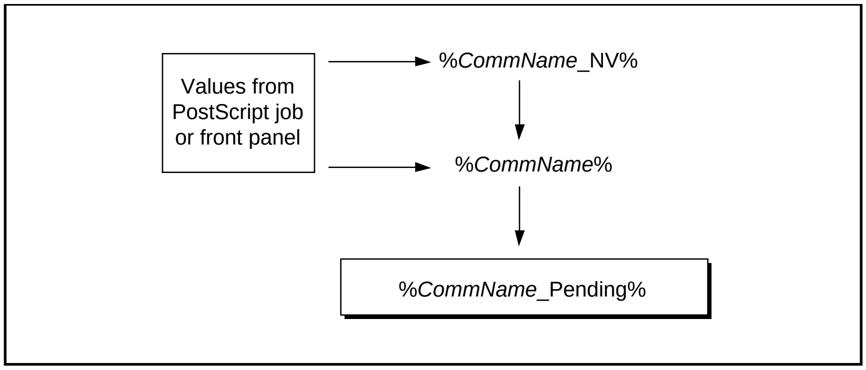

FIGURE 3.3 *Relationship among the Communications parameter sets*

As shown in this diagram, values written to %*CommName*% are written through to %*CommName*_Pending%, and values written to %*CommName*_NV% are written through to %*CommName*% and then to %*CommName*_Pending%. Beyond this, there are several variables:

- 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. Generally, products do not have both a front panel and switches.
- PostScript 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 switches.

The %*CommName*% parameter set, which is in RAM and does not persist when the imaging system is powered off, is used in many cases (but not all) to update the %*CommName*_Pending% set. Thus, on many products, such as those with a front panel but no switches, the %*CommName*% and %*CommName*_Pending% sets always have the same values and appear redundant.

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 imaging system is powered off.

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

Multiple Nonvolatile Parameter Sets (%CommName_NVn%)

It is possible to have multiple nonvolatile parameter sets, named %*CommName*_NV%, %*CommName*_NV2%, %*CommName*_NV3%, and so on. As with a single nonvolatile set, these parameter sets obtain their values by being written to by a PostScript spooler or utility.

Figure 3.4 shows a situation in which there are three nonvolatile parameter sets. Only one of these sets can be active at any given time. In this figure, the active set is %*CommName*_NV2%, as indicated by the switch setting. When the switch is set to this position, or when the product is restarted or powered on 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 job can write to any of the nonvolatile parameter sets, but only values written to %*CommName*_NV2% would 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 values in %*CommName*% and %*CommName*_Pending%.

Predetermined (Hard-Wired) Parameter Values

In addition to the switch settings that indicate which nonvolatile parameter set should be used, other switch settings can affect the hierarchy of parameter sets and cause a predetermined set of communications parameters to be written directly to %*CommName*_Pending%. In Figure 3.5, for example, switch positions 1 and 2 designate two "hard-wired" parameter sets. When the switch is set to position 1, for example, PostScript programs may still write to one of the %*CommName*_NV% parameter sets or to %*CommName*%, but not to %*CommName*_Pending% unless the switch is reset to one of positions 3 through 5.

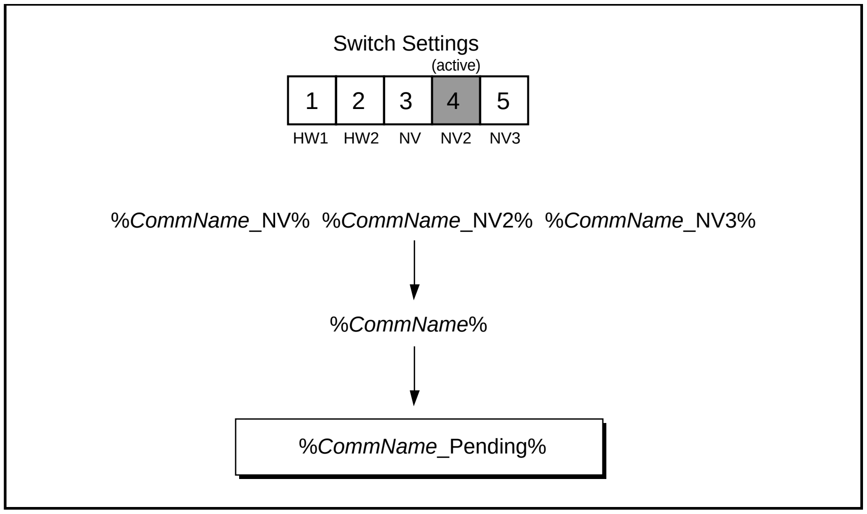

FIGURE 3.4 *Communications parameter sets using NV values*

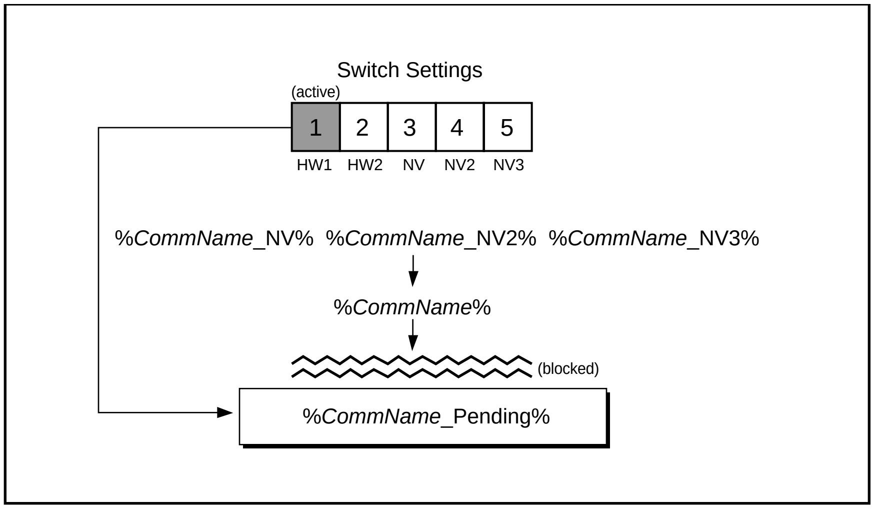

FIGURE 3.5 *Communications parameter sets using "hard-wired" values*

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

Note that reading the *%CommName_NV%* or *%CommName%* set gives 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 it 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 nonvolatile memory or for whether a utility job, front panel, or switch is used to establish communications parameters.

As 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 Communications parameters sets take effect after the current file (containing one or more PostScript jobs) is fully processed by the interpreter and before the next file is read.

Once a file that specifies communications changes has been interpreted, a transition to new settings of the %*CommName*_Pending% set occurs. No additional data should be transferred from the host to the imaging system 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," since a file can contain multiple jobs; see* PLR3, *Section 3.7.7.*

Entries Common to All Communications Parameter Sets

Table 3.3 describes the entries found in all device parameter sets of type Communications.

TABLE 3.3 Device parameters common to all Communications parameter sets

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Communications.
HasNames	boolean	*(Read-only)* A flag specifying whether the device represented by the parameter set (that is, the communication channel) supports named files. This parameter is always *false* in Communications parameter sets.
Enabled	boolean	A flag specifying whether data arriving on the communication channel should be considered as a job to be scheduled for execution by the PostScript interpreter or emulator. If this flag is *true*, arriving data will be scheduled as an executable job; otherwise, 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** occurs in either of the following situations: • Attempting to set **Enabled** to *true* when **On** is *false* in the same parameter set • Attempting to set **Enabled** to *false* in one parameter set when it would result in all channels having **Enabled** set to *false*
On	boolean	A flag specifying whether the communication channel is turned on and able to send and receive data. If this flag is *true*, data transmitted to the channel by a host is buffered and flow control protocols are applied; otherwise, data sent to the channel is lost. A **configurationerror** occurs if an attempt is made to set **On** to *false* in either of the following situations: • When **Enabled** is *true* in the same parameter set • When it would result in all channels having **On** set to *false* If two communications devices share the same physical port and **On** is set to *true* when it is already *true* for another channel associated with this port, 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 a source of jobs to be scheduled, but the channel can be used by a PostScript job to send and receive data by means of the file operators. When the system is powered on, 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 communication channel and force it on, even if this was not the state preserved in nonvolatile memory. An alternative would be to inform the user

via the operator control panel that the product cannot be initialized until the problem is rectified.

DelayedOutputClose	boolean	A flag specifying how the output channel is managed after each job finishes executing. This parameter is set independently for each communication channel. When this flag is *true*:

- An EOF (end-of-file) indication 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 previous jobs using the same output channel have not finished printing, the output will not be sent until those jobs have completed printing and the EOF for each of them has been sent.
- Spontaneous messages, such as imaging system error messages, are sent to the channel whether it is the output channel for the currently executing job or the output channel for previous jobs that have not finished printing.

When the value of **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 previous jobs using the same output channel have not finished printing. (The EOF for each of those jobs will have already been sent.)
- Spontaneous messages are sent to the channel only if it is the output channel for the job currently executing, and 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. Thus, if the serial B channel is used for the output of jobs received on the parallel port, the **DelayedOutputClose** value in the %SerialB% parameter set applies to jobs received on both the serial B port and the parallel port.

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 imaging system error messages, is always sent before the next job begins and before the EOF is sent or a connection closed.*

Interpreter name A name object specifying which interpreter or emulator is to be used to interpret the next incoming job arriving on the communication 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 **Protocol** parameters that can result in a **configurationerror**; see **Protocol** in Table 3.4.

Interpreter (or **Protocol**) can be set without a password if there are no other entries in the dictionary passed to **setdevparams**. If the only additional entry in the dictionary is the password, it is ignored.

The valid values of **Interpreter** are PostScript, AutoSelect, Diablo630, EpsonFX850, HP7475A, LaserJetIII, LaserJetIIP, PCL, and ProprinterXL. The values Diablo630, EpsonFX850, HP7475A, LaserJetIII, LaserJetIIP, and ProprinterXL refer to imaging systems (no longer sold) that offered a page description language (PDL) other than PostScript. The value PCL refers to Hewlett-Packard's Printer Control Language; this value was first introduced when the LaserJet 4 printer was emulated, and it continues to reflect the imaging systems of Hewlett-Packard. The imaging system being emulated is described in an instance of the **PDL** resource category. PCL was first introduced as a value of **Interpreter** when the LaserJet 4 printer was emulated, and it continues to reflect the imaging systems of Hewlett-Packard. For more information on emulation, see Appendix C.

The **Interpreter** value of AutoSelect provides automatic and seamless switching between the available interpreters and emulators based on the input data stream. **Interpreter** should be set to AutoSelect on channels that connect to hosts that alternately send PostScript jobs, raw PCL, and "printscreen" jobs (in the IBM PC–compatible environment). In general, it can be used on any communication channel, but it requires that the underlying communications protocol be one that preserves all incoming data. For serial and parallel communication channels:

- If **Protocol** is set to Raw (the recommended setting when the value of **Interpreter** is AutoSelect), AutoSelect detects interpreter, job, and protocol boundaries and automatically selects the protocol. When it detects that a PostScript job is being received, only Normal and TBCP protocols can be recognized (Binary is not supported).
- If **Protocol** is set to TBCP or Binary, AutoSelect detects interpreter boundaries and job boundaries.
- Attempting to set **Interpreter** to AutoSelect when **Protocol** is Normal (which reserves certain control characters for communications functions) causes a **configurationerror**.

For other communication channels that are binary in nature:

- AutoSelect detects interpreter boundaries and job boundaries.
- The **Interpreter** value of PCL is used only in imaging systems that emulate a LaserJet 4 or later LaserJet printers. For emulations of earlier LaserJet models, the values LaserJetIIP and LaserJetIII are used.

PrinterControl name

A name object that indicates how a host queries and controls the imaging system when accessing it via this communication channel. The valid values of **PrinterControl** are PSPrinter and PJL. If **PrinterControl** is set to PSPrinter:

- The **Interpreter** parameter selects the page description language (PDL) interpreter.
- Imaging system error messages are sent in the usual Adobe fashion (on channels processing jobs, and in %%[…]%% format).
- PJL (Printer Job Language) commands are recognized only if the value of **Interpreter** is AutoSelect or PCL. If **Interpreter** is AutoSelect, the ENTER LANGUAGE and UEL commands are handled and other PJL commands are identified as such and discarded. If **Interpreter** is PCL, only the UEL command is handled.

If **PrinterControl** is set to PJL:

- The selection of the PDL interpreter is controlled by PJL, either by explicit invocation of the PJL ENTER LANGUAGE command or based on the value of the PJL variable PERSONALITY. The value of PERSONALITY can be PCL, POSTSCRIPT, or AUTO, where AUTO provides automatic, seamless switching between interpreters.
- Imaging system 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.
- The “PJL current environment” is used on each invocation of a PDL to set up the initial state.

If **PrinterControl** is set to PSPrinter, a Communications parameter set specifies the interpreter or emulator, error messages conform to PostScript guidelines, and an initial graphics state is established at the start of each job. If **PrinterControl** is set to PJL, the behavior emulates LaserJet communications, which rely on PJL to specify the PDL interpreter needed, the structure of error messages, and the initial graphics state.

PJL is described in the Hewlett-Packard *Printer Job Language Reference Manual*; see the Bibliography for details.

Point-to-Point Communications Parameter Sets

In addition to the entries common to all Communications parameter sets (Table 3.3 above), serial, parallel, and SCSI device parameter sets include the entries listed in the following tables. For more information on these protocols, see Adobe Technical Notes #5009, *Adobe Serial and Parallel Communications Protocols Specification*, and #5010, *Adobe SCSI Input Protocol Specification*.

TABLE 3.4 Additional device parameters specific to the %Serial% (or %Serial*X*%) parameter set

KEY	TYPE	VALUE
Protocol	name	A name object specifying the communications protocol that is used: Normal — Certain control characters are reserved for communications functions, such as end-of-file, software flow control, abort job, and status query, and so cannot be carried as data. This protocol is suitable only for ASCII-encoded PostScript jobs, and not for PostScript jobs containing binary data or for any printer emulation jobs. Raw — All characters are treated as data; there are no reserved characters, and no communications functions are available. Normally, this protocol is suitable for use only with an emulator and not with the PostScript interpreter. However, in products that support an **Interpreter** value of AutoSelect, protocol processing is handled by the AutoSelect facility; therefore, the value of **Protocol** should be Raw when **Interpreter** is set to AutoSelect. TBCP — Tagged Binary Communication Protocol. An encoding scheme allows the full range of 8-bit values to be transmitted as data, while also providing for certain communications functions. In addition, it provides explicit begin-protocol and end-protocol sequences that permit the receiver to switch automatically between Normal and TBCP processing. This protocol is suitable for use with any language (the PostScript interpreter or an emulator). Binary — An encoding scheme allows the full range of 8-bit values to be transmitted as data, while also providing for certain communications functions. This protocol is suitable for use with any language; however, it is obsolete and has been superseded by TBCP. A **configurationerror** occurs if setting either the **Protocol** or the **PrinterControl** parameter would result in the combination of a **Protocol** value

of Normal and a **PrinterControl** value of PJL. A **configurationerror** also occurs if setting **Protocol** or **Interpreter** would result in one of the following combinations when **Enabled** is *true* and **PrinterControl** is PSPrinter:

- A **Protocol** value of Raw and an **Interpreter** value of PostScript
- A **Protocol** value of Normal and an **Interpreter** value other than PostScript

In other words, PostScript 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 automatic selection of interpreters and emulators is chosen, the Normal protocol cannot be used.

Protocol (or **Interpreter**) can be set without a password if there are no other entries in the dictionary passed to **setdevparams**. If the only additional entry in the dictionary is the password, it is ignored.

FlowControl name A name object specifying the serial flow control method used between the host and the device:

Dtr	DTR (data terminal ready) and DSR (data set ready) hardware signals are used by the imaging device and the host, respectively, to indicate to each 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	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 control characters, ETX (end-of-text) and ACK (acknowledgment), are reserved for flow control usage. The protocol is symmetrical for the imaging device and the host. Each sender knows an agreed-upon maximum number of bytes that the other side can receive, and the sender may send up to this number of bytes followed by an ETX. The sender may send more data only after receiving an ACK from the other side.
XonXoff	Used for communicating with PCs (non-UNIX systems). Two control characters, XON and XOFF, are reserved for flow control usage. For all **Protocol** values except Raw, the protocol is symmetric for the imaging device and the host. If one side wants 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 the value of **Protocol** is Raw, XON and XOFF sent from the host to the imaging system are treated as data and not reserved as flow control characters. XON and XOFF sent from the imaging system to the host are treated as flow control characters.
XonXoff2	Used for communicating with UNIX systems; similar to XonXoff 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 wants to send; the interpreter simply drops the data.
RobustXonXoff	Similar to XonXoff except that periodically (typically every second) the interpreter will send the host an XON if it is able to receive data.

Note: *Not all serial channels support all flow control methods.*

Baud	integer	The baud rate on the underlying serial hardware. Normally this parameter can be set to any nonnegative value; however, the underlying serial hardware will round it to the nearest achievable baud rate. 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 valid values.
DataBits	integer	The number of data bits per byte communicated over the channel. Valid values are 7 and 8. If the value is 7, the high bit of a received byte of data is set to 0.
StopBits	integer	The number of stop bits that are transmitted by the serial hardware. Valid values are 1 and 2; the hardware will always be able to receive data transmitted with one or two stop bits.
Parity	name	A name object specifying the parity to be used between the host and the device when **CheckParity** is *true*:

None	Neither the host nor the device should send a parity bit.
Space	The parity bit should always be 0.
Mark	The parity bit should always be 1.
Even	Even parity should be used.
Odd	Odd parity should be 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 (7 or 8), parity bits (0 or 1), and stop bits (1 or 2). Most serial devices do not support 8-bit data with either Space or Mark parity; setting the parameters in this way does not generate a **configurationerror**, but the results of such a configuration are unpredictable.
CheckParity	boolean	A flag specifying whether the device should do parity checking on incoming data. If **CheckParity** is *true* and a parity error occurs, an **ioerror** results. This parameter is ignored if **Parity** specifies None.

TABLE 3.5 Additional device parameters specific to the %Parallel% (or %Parallel*X*%) parameter set

KEY	TYPE	VALUE
Protocol	name	See Table 3.4 above.
Handshake	integer	A code identifying the hardware/software signal interface that is to be used for communications across the parallel interface: 0 Unidirectional communications, commonly used by PCs and PC-compatibles. This is the default. 1 Bidirectional communications, as specified by version 0.6 of the Hewlett-Packard Boise specification. This is an obsolete value; it was superseded by 2, which was later superseded by 6 through 8. 2 IEEE-1284 Draft Specification 1.00. This is an obsolete value, superseded by 6, 7, and 8. 3 Unidirectional, ACK in Busy. 4 Unidirectional, ACK after Busy (Japanese 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. ***Note:*** *The IEEE-1284 Draft Specification 2.00 is the IEEE standard 1284-1994; see the Bibliography for more information.* Attempting to set **Handshake** to one of the unidirectional values (0, 3, 4, or 5) when **OutputDevice** designates a parallel communications device generates a **configurationerror**.
OutputDevice	string	The name of the communications device to be used for %stdout and %stderr—that is, %Serial%, %SerialB%, %SerialC%, and so on, or %Parallel%,

		%ParallelB%, %ParallelC%, and so on. If **OutputDevice** is the empty string, %stdout and %stderr information is routed to a product-specific default output device. Attempting to set **OutputDevice** to a parallel communications device when the value of **Handshake** is one of the unidirectional values (0, 3, 4, or 5) generates a **configurationerror.**
nAckPulseWidth	integer	The width (duration), in nanoseconds, of the *nAck* pulse, rounded to the nearest value that can be achieved. nAck is a signal that originates from the imaging system to the host in the form of a pulse; its meaning depends on which **Handshake** mode is being used or what state the imaging system's device driver is currently in. The nAck pulse width is controlled by the imaging system's support circuitry or the device driver and is normally in the range 500–10,000.
nStrobeExpectedPulseWidth	integer	The expected duration, in nanoseconds, of the *nStrobe* pulse, rounded to the nearest value that can be achieved. nStrobe is a signal that originates from the host to the imaging system in the form of a pulse; its meaning depends on which **Handshake** mode is being used or what state the imaging system's device driver is currently in. The nStrobe pulse width, which may vary from host to host, is normally in the range 750–500,000.

TABLE 3.6 Additional device parameters specific to the %ScsiComm% (or %ScsiComm*X*%) parameter set

KEY	TYPE	VALUE
Bus	string	*(Read-only)* The name of the SCSI bus device parameter set associated with this channel. Valid values are %Scsi%, %ScsiB%, %ScsiC%, and so on.
Filtering	name	A name object indicating whether the input stream needs further filtering before the data can be correctly interpreted as a page description language: None — Passes the data unchanged to the interpreter. InterpreterBased — Filters the input stream as necessary to conform to the language. For example, the data stream may have been sent to the imaging system encoded as a TBCP PostScript job and must be decoded to a normal PostScript job before being passed to the interpreter. See **Protocol** in Table 3.4 on page 33 for a description of TBCP. ***Note:*** *In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to* None*, or communications problems will result.*

Network Communications Parameter Sets

Network communications rely on a combination of Communications and Parameters parameter sets. The Communications parameter sets correspond to the application layer of the OSI model and connect directly to a PostScript interpreter or emulator, which receives its jobs from and sends messages back to the host using the devices associated with these parameter sets.

The network communications parameter sets, described in more detail below, are: %LPR%, %AppSocket%, and %Telnet%, which allow for the use of the TCP/IP protocol over Ethernet; %RemotePrinter% and %PrintServer%, which are associated with the Novell NetWare application layer; and %LAT%, used primarily for communication with terminal servers. This section also describes the device parameter sets for the AppleTalk networking software (LocalTalk, EtherTalk, and TokenTalk), which do not split out the physical layer.

- %LPR%. The UNIX command lpr sends a printer job to a printer system. The lpr service depends on the TCP/IP protocol; TCP port 515 is used for it. Because lpr (or the lpr daemon) is by definition unidirectional, any %stdout or %stderr information is transmitted by means of the Syslog facility (described in Table 3.17 on page 48).
- %AppSocket%. The AppSocket protocol was created to support TranScript® software from Adobe Systems. It provides a more robust interface than %LPR% because it uses bidirectional communications directly. AppSocket can also be used by drivers other than TranScript and for transmitting data other than PostScript jobs. For more information about AppSocket, contact Adobe Developer Support.
- %Telnet%. Telnet, which is a TCP/IP network service, gives network users interactive access to and exclusive use of the PostScript interpreter or emulator. TCP port 23 is used for Telnet.
- %RemotePrinter%. The Novell remote printer application is managed by a Novell print server and takes print jobs downloaded from a print server. A remote imaging system may be shared by many print servers.
- %PrintServer%. The Novell print server application 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.
- %LAT%. The LAT (Local Area Transport) 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 set of imaging systems on a local area network.

- %LocalTalk%. LocalTalk is the name given to AppleTalk running over LocalTalk or PhoneNET networks.
- %EtherTalk%. EtherTalk is the name given to AppleTalk running over Ethernet networks.
- %TokenTalk%. TokenTalk is the name given to AppleTalk running over Token-Ring networks.

In addition to the entries common to all Communications parameter sets (Table 3.3 on page 29), the above device parameter sets include the entries listed in the following tables, with one exception: there are no additional entries in the %Telnet% (or %Telnet*X*%) parameter set.

TABLE 3.7 Additional device parameters specific to the %LPR% (or %LPR*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.
PrintHost	string	A string of up to 63 characters specifying at most two IP mask/address pairs. The mask is applied to the IP address to specify which hosts are allowed to make lpr connections. The format of this string is [*mask1/*]*address1* [[*mask2/*]*address2*] where everything in square brackets is optional. The format of an IP mask or address is *N.N.N.N*, where each *N* is a decimal number in the range 0–255. If an address is specified without a corresponding mask, a mask of 255.255.255.255 is assumed. For example, 138.46.25.37 is a mask/address pair in which the mask is not specified, whereas 255.255.255.0/138.46.25.37 255.255.255.0/138.46.24.38 is two mask/address pairs, each specifying a mask. Specifying an invalid IP address, such as 0.0.0.0, 127.0.0.0, or *N.N.N.*255, will result in a **rangecheck** error.
SendWindowSize	integer	The send window size (the number of bytes that the imaging system will transmit in a data packet). This parameter provides a means of tuning the

		code for optimal throughput. Its setting is applied at imaging system startup, 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's value in order to accommodate the host's expectations.
		The window size specified here overrides any request for this parameter in the associated sets of type Parameters—for example, %TCP%. The valid range of values for this parameter is 1024–65535.
ReceiveWindowSize	integer	The receive window size (the number of bytes that the host will transmit in a data packet). See **SendWindowSize** for additional information.

TABLE 3.8 Additional device parameters specific to the %AppSocket% (or %AppSocket*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.
PrintHost	string	See Table 3.7 above.
SendWindowSize	integer	See Table 3.7 above.
ReceiveWindowSize	integer	See Table 3.7 above.
ControlPortNumber	integer	The port number of a port used by the unit for handshaking between the host and the unit while setting up a session. A session with the imaging system prevents other hosts from being able to interrupt the imaging system to run other jobs. Communication is via TCP, not UDP. The suggested default value is 9101. **ControlPortNumber** can refer to the same port as **StatusPortNumber**, but a **configurationerror** will occur if it refers to the same port as **DataPortNumber**.
		A **configurationerror** will also occur if the same value is specified for this parameter in more than one parameter set (such as %AppSocket% and %AppSocketB%); however, if port numbers are assigned by means other than parameter sets, no error occurs if any of those port numbers is the same as the value specified for this parameter. (This applies to **StatusPortNumber** and **DataPortNumber** as well.)
StatusPortNumber	integer	The port number of a port used by the unit for sending status information back to the host. When TCP/IP is used, communication is via UDP, not the TCP transport layer. The suggested default value is 9101. A different port number may be used to avoid a conflict if another unit on the network is already using 9101.

		StatusPortNumber can refer to the same port as **ControlPortNumber**, but a **configurationerror** will occur if it refers to the same port as **DataPortNumber**.
DataPortNumber	integer	The port number of a bidirectional port for transmission of printer jobs. The suggested default value is 9100. A different port number may be used to avoid a conflict if another unit on the network is already using 9100. If **DataPortNumber** refers to the same port as either **ControlPortNumber** or **StatusPortNumber**, a **configurationerror** will occur.

TABLE 3.9 Additional device parameter specific to the %RemotePrinter% (or %RemotePrinter*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.

TABLE 3.10 Additional device parameters specific to the %PrintServer% (or %PrintServer*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.
LoginPassword	string	A string of up to 32 characters specifying 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 **LoginPassword** when queried is always the string INVALID; attempts to set **LoginPassword** to this string will be ignored.
PreferredServer	string	The name of the file server that the print server attempts to attach to in order to service print queues. The validity of the **PreferredServer** name is not checked; the value is passed directly to the nearest file server for routing request if the file server specified by **PreferredServer** does not respond. Novell 3.x file server names are limited to 47 characters; spaces and the characters " * + , \ / | ; : = < > ? [ ] are not allowed. Novell 4.x file server names can contain up to 64 characters; spaces are allowed but not desirable. Novell NetWare 4.x converts spaces to underscores. (Strings that contain spaces may not work on Novell NetWare 3.x.)

TABLE 3.11 Additional device parameters specific to the %LAT% (or %LAT*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.
Physical	string	*(Read-only)* The physical layer over which LAT is accessed; a device parameter set corresponding to a physical communications medium, such as %EthernetPhysical%. This parameter can associate one network layer parameter set with one and only one physical layer parameter set.
Groups	string	The groups allowed to access this printer device. A group is specified by an integer in the range 0–255. A range of groups is designated with a hyphen, and multiple groups or ranges are separated by spaces. For example, the string 3-8 145-160 200 designates two ranges and one additional group. The default value is the empty string, which gives all groups access.
KeepaliveTimer	integer	The interval (in seconds from 0 to 180) at which the circuit layer exchanges keep-alive messages to maintain circuits when no session traffic is present.
MulticastTimer	integer	The interval (in seconds from 0 to 180) at which directory service advertisements (SAs) are multicast to advertise the printer's service.
RetransmitTimer	integer	The interval (in seconds from 0 to 10) at which the circuit layer retransmits unacknowledged messages.
RetransmitLimit	integer	The number of retransmissions (from 4 to 120) that will be attempted before a circuit is declared dead.

TABLE 3.12 Additional device parameters specific to the %LocalTalk% (or %LocalTalk*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.
NodeID	integer	*(Read-only)* The local network address of the device. Valid addresses are in the range 128–254. A value of 0 indicates that the address has not been established. The value of this parameter is used as an address hint when addresses are first established as part of the AppleTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the AppleTalk device.
LocalTalkType	string	The *type* component of the AppleTalk entity name. The entity name consists of the three components *zone*, *type*, and *object*, each a string of up to 32 characters. The *zone* component is set to the wildcard character (*), and the *object* component is set to the value of the **PrinterName** system parameter.

The conventional value of **LocalTalkType** for any PostScript-capable printer is LaserWriter. Imaging systems having this type are automatically available for direct use by any host. A spooler or other system administration program can change the value of **LocalTalkType** to make the imaging system inaccessible for direct use.

The **appletalktype** compatibility value will reflect a change to the **LocalTalkType** parameter. If the imaging system also supports EtherTalk or TokenTalk communications, setting **LocalTalkType** will set the **EtherTalkType** or **TokenTalkType** parameter to the same value.

TABLE 3.13 Additional device parameters specific to the %EtherTalk% (or %EtherTalk*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.
NodeID	integer	*(Read-only)* The local network address of the device. Valid addresses are in the range 1–254. A value of 0 indicates that the address has not been established. The value of this parameter is used as an address hint when addresses are first established as part of the AppleTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the AppleTalk device.
EthernetAddress	string	*(Read-only)* The Ethernet address of the imaging system. The format is *xx:xx:xx:xx:xx:xx*, where each *x* is a hexadecimal digit (0–9 or either A–F or a–f).
EtherTalkType	string	The *type* component of the EtherTalk entity name. The entity name consists of the three components *zone*, *type*, and *object*, each a string of up to 32 characters. The *zone* component is set to the value of the **EtherTalkZone** parameter, and the *object* component is set to the value of the **PrinterName** system parameter. The **appletalktype** compatibility value will reflect a change to the **EtherTalkType** parameter. If the imaging system also supports LocalTalk or TokenTalk communications, setting the **EtherTalkType** string will set the **LocalTalkType** or **TokenTalkType** parameter to the same value.
EtherTalkZone	string	The *zone* portion of the EtherTalk entity name. See **EtherTalkType**.

TABLE 3.14 Additional device parameters specific to the %TokenTalk% (or %TokenTalk*X*%) parameter set

KEY	TYPE	VALUE
Filtering	name	See Table 3.6 on page 37.

NodeID	integer	*(Read-only)* See Tab le 3.13 above.
Address	string	*(Read-only)* The TokenRing address of the imaging system. The format is *xx:xx:xx:xx:xx:xx*, where each *x* is a hexadecimal digit (0–9 or either A–F or a–f).
Bridging	name	A name object specifying how to do bridging on the TokenRing network: Adaptive — The software will automatically recognize the routing style and respond in kind (either as a one-time determination or each time it processes a connection). This is the default value. Transparent — The entire "universe" is one large single ring structure and all identities are unique. SourceRoute — Routing is done by specifying an explicit path, including the ring identification, or Routing Information Field (RIF).
TokenTalkType	string	The *type* component of the TokenTalk entity name. The entity name consists of the three components *zone*, *type*, and *object*, each a string of up to 32 characters. The *zone* component is set to the value of the **Zone** parameter, and the *object* component is set to the value of the **PrinterName** system parameter. The **appletalktype** compatibility value will reflect a change to the **TokenTalkType** parameter. If the imaging system also supports LocalTalk or EtherTalk communications, setting the **TokenTalkType** parameter will set the **LocalTalkType** or **EtherTalkType** parameter to the same value.
Zone	string	The *zone* component of the TokenTalk entity name. See **TokenTalkType**.

3.3.4 Parameters Parameter Sets

Parameter sets of type Parameters are used to configure the following:

- The SNMP and Syslog network services
- The lower layers of network communications
- The SCSI bus and the IDE bus
- The print engine, the operator console, and the calendar

For network communications, these parameter sets correspond to the transport, network, data link, and physical layers of the OSI model, as shown in Figure 3.1 on page 20. (The PostScript interpreter or emulator receives its jobs from the application layer.) For each parameter set associated with an implementation of the network layer, there must be a parameter set associated with the data link and physical layers (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 and 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.

The Parameters parameter sets are described in more detail below. Except where indicated otherwise in these descriptions, changes to parameters in the parameter set do not take effect until the imaging system is reinitialized. Note that as with Communications parameter sets, when there are multiple instances of a particular Parameters parameter set, the naming convention for instances beyond the first is to add the letter B for the second instance, C for the third, and so on—for example, %Scsi% and %ScsiB% if more than one SCSI bus is present.

- %SNMP%. SNMP (Simple Network Management Protocol) allows 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). The parameters listed for the %SNMP% parameter set in this *Supplement* are only those that need to be accessible from the PostScript language as opposed to the MIB (or MIBs) the device may have. These are the only parameters that are changeable from an environment separate from SNMP (the network side). The rules governing when changes to parameters in this parameter set take effect are included in the description of each parameter.

  More information about SNMP is available in Internet Engineering Task Force Request for Comments (IETF RFC) 1157.

- %Syslog%. Syslog is a logging facility that sends log messages back to a UNIX host. Most of the messages contain network-specific information, but they may include any other pertinent information the unit wants to convey. Communication is via the UDP transport layer.

- %TCP%. TCP is the transport layer responsible for reliable data transfer; it is a connection-oriented IP-based protocol that guarantees message delivery and reception. Any packet that is lost will be retransmitted.

- %UDP%. UDP is the connectionless IP-based protocol. When UDP is used 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.

- %IP%. IP 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 messages.
- %SPX%. SPX is the Novell NetWare connection-oriented protocol. When SPX is used to communicate with a peer host, handshaking takes place before the connection is ready. The delivery of SPX packets is 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 (that is, the ability to temporarily change the data packet size) for flow control.
- %IPX%. IPX is the Novell NetWare connectionless (or datagram) protocol. When IPX is used 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 uses IPX.
- %EthernetPhysical% and %TokenRingPhysical%. These correspond to a physical Ethernet or TokenRing connector, respectively, along with its associated hardware and the data link layer software that handles events from this device.
- %Scsi% and %Ide%. These are used to configure the SCSI bus or IDE bus, respectively. The parameter set is always present in an imaging system that has that type of bus, even if no devices are present on the bus.
- %Engine%. This parameter set is used to configure the print engine. The rules governing when changes to parameters in this parameter set take effect are product-dependent, but typically the effect is immediate.
- %Console%. This parameter set provides a means of setting and controlling characteristics of the operator console of an output device that includes the PostScript language. The parameters currently defined in this set provide an extensible way of selecting the natural language (for example, English or Japanese) in which information will be displayed on the operator console. The rules governing when changes to parameters in this parameter set take effect are product-dependent, but typically the effect is immediate.
- %Calendar%. A printer has a battery-powered time-of-day clock. This clock must be set initially, and then twice a year to follow daylight saving time (if applicable). The string %Calendar% identifies the calendar device, and the entries in the dictionary describe the local date and time. Changes to parameters in this parameter set take effect immediately.

Required entries for the parameter sets of type Parameters are described in the tables that follow in this section. Some of these parameter sets include the specification of a *node address*, a unique address for a node on a network. The form of the node address depends on the protocol being used to communicate with the node. Table 3.15 lists the various forms a node address can take.

TABLE 3.15 Forms of a node address

PROTOCOL	ADDRESS FORM	DESCRIPTION
Novell IPX/SPX	*XXXXXXXX*:*xxxxxxxxxxxx*	Each *X* or *x* is a hexadecimal digit (0–9 and either A–F or a–f). The 8-digit *XXXXXXXX* is the network part of the address, and the 10-digit *xxxxxxxxxxxx* is the MAC (media access control) part, known as the *Novell node number.*
AppleTalk DDP	*N.N.n*	Each *N* or *n* is a decimal number in the range 0–255. *N.N* is the network part of the address, and *n* is the node ID.
TCP/IP	*N.N.N.N*	An *IP address*. Each *N* is a decimal number in the range 0–255.

TABLE 3.16 Device parameters in the %SNMP% (or %SNMP*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
SysName	string	A string of up to 32 characters specifying the name of the imaging system (expected by SNMP). Changes to this parameter take effect immediately.
SysLocation	string	A string of up to 32 characters specifying the location of the imaging system. Changes to this parameter take effect immediately.
SysContact	string	A string of up to 32 characters that traditionally specifies the name and either the phone number or the address of the person responsible for the imaging system. Changes to this parameter take effect immediately.
PrivateHost	string	A string of up to 49 characters specifying one node address per protocol—the address of a host that is able to set those SNMP variables that can be set. The various forms of a node address are listed in Table 3.15 above. If there is more than one such address, they are separated by spaces. The empty string, indicating that no host has access, is the usual default value; the imaging system will therefore need to have this parameter set explicitly with the **setdevparams** operator before SNMP can be used. Changes to this parameter do not take effect until the imaging system is reinitialized.

TrapHost	string	A list of one or more specifications of the following format for each host that is able to receive traps (asynchronous messages, often error conditions, that are reported to a network management station): *nodeAddress/community* If there is more than one such specification, they are separated by spaces. The various forms of a node address are listed in Table 3.15 above. The value of *community* (which is case-insensitive) specifies the type of host, with values like public, proxy, private, regional, and core. For example, the value 130.248.224.46/public specifies an IP address for a trap host node in a public community. The empty string, indicating that no traps are being sent to a host, is the usual default value; the imaging system will therefore need to have this parameter set explicitly with the **setdevparams** operator before the trap host facility can be used. Changes to this parameter do not take effect until the imaging system is reinitialized.

TABLE 3.17 Device parameters in the %Syslog% (or %Syslog*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
LogHost	string	The IP address for a host that receives Syslog messages from the unit. For the format of an IP address, see Table 3.15 on page 47. The empty string indicates that no Syslog messages are to be sent by the unit. Specifying an IP address equal to 0.0.0.0 or *N.N.N.*255 will result in a **rangecheck** error.
LogPriority	integer	The lowest priority of logging messages that are to be sent on to the Syslog host. All messages of this priority or higher (where smaller values designate higher priority) are sent. The priorities, listed below from highest to lowest, follow the BSD UNIX and SunOS™ convention. 0 Messages indicating that the unit is no longer usable 1 Messages indicating that immediate action is needed on the part of a system administrator 2 Critical error messages 3 Noncritical error messages 4 Warning messages 5 Normal but significant conditions

6	Informational messages
7	Debugging messages

TABLE 3.18 Device parameters in the %TCP% (or %TCP*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the TCP protocol is activated at imaging system startup.
SendWindowSize	integer	See Table 3.7 on page 39.
ReceiveWindowSize	integer	See Table 3.7 on page 39.

TABLE 3.19 Device parameters in the %UDP% (or %UDP*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the UDP protocol is activated at imaging system startup.
Checksum	boolean	A flag specifying whether checksum values will be inserted in outgoing packets formed by the software. The default value should be *true*.

TABLE 3.20 Device parameters in the %IP% (or %IP*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the IP protocol is activated at imaging system startup.
Physical	string	*(Read-only)* The name of the physical layer over which the protocol is accessed. The string designates a device parameter set corresponding to a physical communications medium, such as %EthernetPhysical%. This parameter can associate a network layer parameter set with one and only one physical layer parameter set.
TransmitEncapsulation	name	A name object specifying the transmit encapsulation type:

		SNAP 802.2 or 802.5 header with a SNAP header. This is the default value when **Physical** is %TokenRingPhysical%. DIX *(Ethernet only)* Ethernet II header. This should be the default value when **Physical** is %EthernetPhysical%. ***Note:*** *These values were introduced to eliminate dependencies on the type of connection used (token-ring or Ethernet). The valid values in version 2016 were 802.3-2-SNAP, 802.5-2-SNAP, and DIX.*
IPAddressDynamic	boolean	A flag controlling how the **IPAddress** and related parameters (**Broadcast-Address**, **NetworkMask**, and **GatewayAddress**) are assigned. If **IPAddress-Dynamic** is *true*, these parameters are obtained automatically from the network during imaging system startup, using the BOOTP or RARP protocol. If **IPAddressDynamic** is *false*, these parameters are set explicitly using **setdevparams**. A change to the value of **IPAddressDynamic** takes effect at the next imaging system initialization. An invocation of **setdevparams** that changes **IPAddress** or related parameters must also set **IPAddressDynamic** to *false*.
IPAddress	string	The IP address of the imaging system. For the format of an IP address, see Table 3.15 on page 47. This address is mapped to the imaging system's physical layer address as specified by the **Physical** parameter (for example, **EthernetAddress** if **Physical** is %EthernetPhysical%). The default value is the empty string, which means that the IP protocol layer is not active. The value of **IPAddress** can be set explicitly at any time; however, it will take effect only at the next imaging system initialization, and only if the **IPAddressDynamic** parameter is set to *false* in the same **setdevparams** call that sets the value of **IPAddress**. Specifying an IP address equal to 127.*N.N.N*, *N.N.N*.0, *N.N.N*.255, or any address whose first field is in the range 224–255 will result in a **rangecheck** error.
NetworkMask	string	A mask indicating which fields of the value of the **IPAddress** parameter designate the network portion of the IP address and which designate the node portion. The format is the same as that of an IP address, as described in Table 3.15 on page 47. For example, 255.255.255.0 is the **NetworkMask** value for a class B network with subnets. **NetworkMask** is used to determine whether a certain IP address is on the same network as the imaging system. The value of **NetworkMask** can be set explicitly at any time; however, it will take effect only at the next imaging system initialization, and only if the **IPAddressDynamic** parameter is set to *false* in the same **setdevparams** call that sets the value of **NetworkMask**.

		If the value of **NetworkMask** is not valid with respect to the value of **IPAddress**, it will be changed to a value that is valid, with no warning to the user. For example, if a class A network mask is given with a class B IP address, **NetworkMask** will be changed to the default class B network mask. The default masks are 255.0.0.0, 255.255.0.0, or 255.255.255.0 for class A, B, and C, respectively; no subnets are accounted for in these masks.
BroadcastAddress	string	The broadcast address mask used when broadcasting messages to the local network. The format is the same as that of an IP address, as described in Table 3.15 on page 47.
		The value of **BroadcastAddress** can be set explicitly at any time; however, it will take effect only at the next imaging system initialization, and only if the **IPAddressDynamic** parameter is set to *false* in the same **setdevparams** call that sets the value of **BroadcastAddress**.
		If the value of **BroadcastAddress** is not valid with respect to the values of **IPAddress** and **NetworkMask**, it will be changed to a value that is valid, with no warning to the user. For example, suppose **IPAddress** is 134.14.15.16 and **BroadcastAddress** is 134.14.255.255; if the user changes **IPAddress** to 134.15.15.16 without explicitly changing **BroadcastAddress**, **BroadcastAddress** will be changed to 134.15.255.255.
GatewayAddress	string	The *destinationAddress*/*gatewayAddress* pairs to other networks, where the addresses are IP addresses separated by a slash and multiple address pairs are separated by spaces. The number of address pairs depends on the product. For the format of an IP address, see Table 3.15 on page 47. Loop-back addresses (127.*N*.*N*.*N*) and broadcast addresses (*N*.*N*.*N*.255) will result in a **rangecheck** error.
		destinationAddress is the network address. Class A network addresses require the first field to be nonzero; the other fields may be zero (for a network with no subnets) or may contain subnet address information. Class B or class C network addresses require the first two or three fields, respectively, to be nonzero.
		A network address of 0.0.0.0 is a special value that is used if none of the specified network addresses matches the desired target IP address. If the same destination address is specified in more than one address pair, all but the last such pair will be ignored.
		The empty string specifies dynamic routing, which means that a Routing Information Protocol (RIP) request to the network is used to gather routing information. If dynamic routing is available in the product, this should be the default value.

The value of **GatewayAddress** can be set at any time; however, it will take effect only at the next imaging system initialization, and only if the **IPAddressDynamic** parameter is set to *false* in the same **setdevparams** call that sets the value of **GatewayAddress**. If **IPAddressDynamic** is *true*, **GatewayAddress** should be set to appropriate valid values (and **IPAddressDynamic** set to *false*, as required), since dynamic routing is not always reliable when the **IPAddress** value is received via RARP.

TABLE 3.21 Device parameters in the %SPX% (or %SPX*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the SPX protocol is activated at imaging system startup.
ReceiveWindowSize	integer	See Table 3.7 on page 39. The valid range of values for this parameter is 1024–59392.

TABLE 3.22 Device parameters in the %IPX% (or %IPX*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the IPX protocol is activated at imaging system startup.
Checksum	boolean	A flag specifying whether checksum values will be inserted in outgoing packets formed by the software. The default value should be *true*.
Physical	string	*(Read-only)* See Table 3.20 on page 49.
TransmitEncapsulation	name	A name object specifying the transmit encapsulation type: NO_SNAP — 802.2 or 802.5 header without a SNAP header. This should be the default value when **Physical** is %TokenRingPhysical%. SNAP — 802.2 or 802.5 header with a SNAP header. 802.3 — *(Ethernet only)* 802.3 header. This should be the default value when **Physical** is %EthernetPhysical%. DIX — *(Ethernet only)* Ethernet II header.

Adaptive	Indicates that, by the nature of the interaction between host and imaging system, the encapsulation format to use in responses to the host can be derived at imaging system startup.

The value of this parameter is checked solely for validity; it is not checked for applicability.

Note: *These values were introduced to eliminate dependencies on the type of connection used. The valid values in version 2016 were 802.3-2, 802.3-X, 802.3-2-SNAP, 802.5-2-SNAP.*

The values of **TransmitEncapsulation** and **Physical** for the various Novell frame types are as follows:

Novell frame type	**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%

NetworkAddress	string	*(Read-only)* The address of the network in which the unit is located. The format is *XXXXXXXX*, where each *X* is a hexadecimal digit. The concatenation of the **NetworkAddress** value and the Novell node number forms the node address that will uniquely identify the unit on the network. The **NetworkAddress** value is obtained from the Novell file server on the local network at imaging system startup. The Novell node number is derived from the MAC (media access control) address of the networking media. For the Ethernet, the Novell node number is the value of the **EthernetAddress** parameter of the %EthernetPhysical% set.
HopCount	integer	The maximum number of routers the print server will go through in trying to attach to file servers while looking for print queues to service. The preferred value is the smallest value needed to reach all of the imaging system's file servers. A count of 15 is defined as trying to reach all reachable servers. Specifying a negative value or a value larger than 15 will set the count to 15. If the **PreferredServer** parameter is set in the %PrintServer% parameter set, the **HopCount** parameter is ignored unless the server specified by **PreferredServer** is unreachable.

TABLE 3.23 Device parameters in the %EthernetPhysical% (or %EthernetPhysical*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the Ethernet channel is enabled at imaging system startup.
EthernetAddress	string	*(Read-only)* The Ethernet address of the unit. The format is *xx:xx:xx:xx:xx:xx*, where each *x* is a hexadecimal digit. ***Note:*** *When Novell NetWare is used, the Ethernet address (minus its colons) is also the Novell node number.*
ConnectorType	name	*(Read-only)* A name object specifying which Ethernet connector type is being used. Valid values are RJ45, BNC, AUI, and AAUI.
Name	string	*(Read-only)* A string of up to 16 characters specifying a mnemonic name for the interface being used—for example, le0 for "lance chip interface unit 0" or so0 for "sonic chip interface unit 0."

TABLE 3.24 Device parameters in the %TokenRingPhysical% (or %TokenRingPhysical*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
On	boolean	A flag specifying whether the TokenRing channel is enabled at imaging system startup.
Address	string	*(Read-only)* The TokenRing address of the unit. The format is *xx:xx:xx:xx:xx:xx*, where each *x* is a hexadecimal digit.
Bridging	name	See Table 3.14 on page 43.
ConnectorType	name	*(Read-only)* A name object specifying which TokenRing connector type is being used. Valid values are RJ45, DB9, and MAU.
Name	string	*(Read-only)* See Table 3.23 above.
Speed	integer	The speed at which the ring is operated, in megabits per second. Valid values are 4 and 16.

TABLE 3.25 Device parameters in the %Scsi% (or %Scsi*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
CheckParity	boolean	A flag specifying whether parity on the SCSI bus is to be checked. The default value is usually *true*. ***Note:*** *Setting* ***CheckParity*** *to* true *on products that do not support parity checking is inadvisable.*
BootDelay	integer	*(Only when disks are present on the bus)* The number of seconds the disk I/O driver should wait during imaging system startup for the disk attached to the imaging system 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. This parameter should be set in accordance with the characteristics of the disk.
InitiatorId	integer	*(Only when disks are present on the bus)* The SCSI bus address used by the imaging system when it serves as initiator. The valid range of values is 0–7. The default value is usually 7.
TargetId	integer	*(Only when a %ScsiComm% communication channel is present on the bus)* The SCSI bus address reserved by the imaging system for use as the %ScsiComm% communication channel. This address may be the same as the **InitiatorId** address. The valid range of values is 0–7.
Poll	integer	*(Only when disks are present on the bus)* A bit mask in the range 0–254 indicating which addresses on the SCSI bus should be polled by the imaging system when it looks for disks during system initialization. Bits are numbered from right to left, beginning with 0 for the low-order bit. Bits set to 1 mean that polling should occur; bits set to 0 mean "do not poll." For example, a 1 in bit 0 means poll for %disk0%. Any bits in the mask that correspond to addresses used as the following should be set to 0: the imaging system's **InitiatorId** or **TargetId** address; the **InitiatorId** address of other hosts on the bus; or the **TargetId** address of devices belonging to other hosts on the bus. The value of **Poll** cannot be 0, since at least the bit corresponding to the **InitiatorId** address should be set to 1. The default value is usually 127 (16#7F). If a bit is set to poll an address that should not be polled, anomalies may occur on the bus. Specifically, if the bit is set to poll the address corresponding to the imaging system's **InitiatorId** or **TargetId** address, a **configurationerror** will occur.

TABLE 3.26 Device parameters in the %Ide% (or %Ide*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
BootDelay	integer	*(Only when disks are present on the bus)* See Table 3.25 above.
Poll	integer	*(Only when disks are present on the bus)* A bit mask in the range 0–3 indicating which addresses on the IDE bus should be polled by the imaging system when it looks for disks during system initialization. For example, a 1 in bit 0 means poll for %disk0%.

TABLE 3.27 Device parameters in the %Engine% (or %Engine*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
BSizeStandard	name	A name object that determines the physical dimensions of the paper when B4 or B5 paper is selected. (Note that "default unit" in the following descriptions means 1⁄72 inch.) ISO The International Organization for Standardization, also known as ISO, defines the "metric" paper sizes (A3, A4, B3, B4, B5, and so on), which are used in Europe and much of the rest of the world. The dimensions for the B4 and B5 paper sizes as defined by ISO are as follows: B4 250 × 353 mm or 709 × 1001 default units B5 176 × 250 mm or 499 × 709 default units JIS The Japanese Industrial Standards Committee specifies standards for use in Japan. Japan uses the standard "A" paper sizes but observes a slightly different definition of the "B" paper sizes. The dimensions for the B4 and B5 paper sizes as defined by JIS are as follows: B4 257 × 364 mm or 729 × 1032 default units B5 182 × 257 mm or 516 × 729 default units
Darkness	number	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. Valid values range from 0.0 to 1.0, where 0.0 means minimum darkness and 1.0 means maximum darkness. This parameter is

		provided in some products whose marking hardware allows software control of colorant application.
DarknessBlack	number	The overall lightness or darkness of the black color on a rendered page produced on a device with multiple toner stations. See **Darkness** for further information.
DarknessCyan	number	The overall lightness or darkness of the cyan color on a rendered page. See **Darkness** for further information.
DarknessMagenta	number	The overall lightness or darkness of the magenta color on a rendered page. See **Darkness** for further information.
DarknessYellow	number	The overall lightness or darkness of the yellow color on a rendered page. See **Darkness** for further information.
PageCount	integer	The number of pages fed by the print engine. This count includes all the pages that were printed successfully as well as those that were jammed or otherwise spoiled. The value of this parameter is determined by querying the engine.
TimeToStandby	integer	The number of minutes after which the print 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 allowable values for this parameter is typically 0–720 (where 0 never lets the engine go into standby mode). If an invalid value is specified, it will be rounded to the nearest allowable value.

TABLE 3.28 Device parameters in the %Console% (or %Console*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
CharSet	name	A name object specifying either a standard character set or a vendor-specific character set. If a standard set, the name will designate the standard or, if applicable, a variant of the standard (such as ISO-646-IRV, a variant of ASCII). The valid standard character set names are as follows:

ASCII	Basic ASCII (7-bit); currency symbol is $
ISO-646-IRV	Same as ASCII except that an international currency symbol replaces $
ISO-8859-1	The ISO 8-bit Latin1 character set

		Adobe-Japan1-0	The CID-keyed Japanese character collection
			If the name is vendor-specific, it should designate the vendor and the identification of the character set used by that vendor—for example, IBM-Codepage-550.
Language	name	A name object specifying the natural language in which information will be displayed, with a 2-character language code from the ISO 639 standard. Table 3.29 lists a selection of these codes; the complete list may be obtained from the International Organization for Standardization (see the Bibliography).	
Country	name	A name object that indirectly specifies the dialect of the language by specifying the country in which the dialect is used. The country is indicated by a 2-character country code from the ISO 3166 standard. Table 3.30 lists a selection of these codes; the complete list may be obtained from the International Organization for Standardization (see the Bibliography).	

TABLE 3.29 Selected ISO 639 language codes

CODE	LANGUAGE	CODE	LANGUAGE
CS	Czech	JA	Japanese
DA	Danish	KO	Korean
DE	German	NL	Dutch
EL	Greek	NO	Norwegian
EN	English	PL	Polish
FI	Finnish	PT	Portuguese
FR	French	RU	Russian
GA	Irish	SK	Slovak
HU	Hungarian	SV	Swedish
IT	Italian	ZH	Chinese
IW	Hebrew		

TABLE 3.30 Selected ISO 3166 country codes

CODE	COUNTRY	CODE	COUNTRY
AR	Argentina	IN	India
AU	Australia	IT	Italy
BE	Belgium	JP	Japan
BO	Bolivia	LU	Luxembourg
BR	Brazil	MX	Mexico
CA	Canada	NL	Netherlands (Holland)
CL	Chile	NO	Norway
CN	China	NZ	New Zealand
CO	Colombia	PA	Panama
DE	Germany	PE	Peru
DK	Denmark	PH	Philippines
EC	Ecuador	PK	Pakistan
ES	Spain	PL	Poland
FI	Finland	PT	Portugal
FR	France	PY	Paraguay
GB	United Kingdom	SA	Saudi Arabia
GR	Greece	SE	Sweden
HU	Hungary	TW	Taiwan
ID	Indonesia	US	United States
IE	Ireland	VE	Venezuela
IL	Israel	ZA	South Africa

TABLE 3.31 **Device parameters in the %Calendar% (or %Calendar*X*%) parameter set**

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Parameters.
Running	boolean	A flag specifying whether the clock should be running. Setting the date and time requires turning on the clock (by setting this parameter to *true*) and simultaneously setting the date and time parameters listed below. When queried, this parameter specifies whether the clock is currently running. If the clock is not running (for whatever reason) or is assumed to be inaccurate (as discovered by internal checks of the clock operation), the **Year** parameter returns 0 and the other date and time parameters return the minimum values in their respective ranges.
Year	integer	The year, in the range 1980–2079, or 0 if queried when the clock is not running or is assumed to be inaccurate.
Month	integer	The month, in the range 1–12.
Day	integer	The day of the month, in the range 1–31.
Hour	integer	The hour, in the range 0–23.
Minute	integer	The minute, in the range 0–59.
Second	integer	The second, in the range 0–59.

3.3.5 FileSystem Parameter Sets

The four commonly used parameter sets of type FileSystem are:

- %disk*n*% (where *n* starts from 0 for the first instance)
- %cartridge% (or %cartridge*n*%, where *n* starts from 1 for the second instance)
- %rom% (or %rom*n*%), denoting a cartridge device that is nonremovable and nonwriteable—that is, some form of ROM
- %ram% (or %ram*n*%), denoting a file system that is writeable and stored in some form of RAM

The memory used by %ram% competes with other uses of memory, such as PostScript virtual memory, and explicit use of %ram% to store files competes with other uses of %ram%, such as storing data for **ReusableStreamDecode** filters.

***Note:** %ram% is required on all version 3010 products. As of version 3011, the access attributes r+, w+, and a+ have been implemented for %ram%. For a description of these attributes, see* PLR3, *pp. 79–80.*

Workstation-based imaging systems that use the native file system have a separate device, os, that provides direct access to that file system; a FileSystem parameter set, %os%, is associated with this device. Most of the parameters in this set are read-only; a minimal set of parameters is provided primarily for consistency with other types of file systems.

As shown in the list above, digit suffixes are used to distinguish between multiple instances of parameter sets for storage devices, and only a disk device parameter set uses the suffix 0. Letter suffixes are generally used only to distinguish between multiple instances of other kinds of FileSystem parameter sets, such as %os%.

Required entries in FileSystem parameter sets are listed in the tables below. Note that in these tables, *read-only* refers only to access by language operators (for example, **setdevparams** and **currentdevparams**). The value of a read-only parameter can change, but not as a result of invoking **setdevparams**. Changes to parameters in FileSystem parameter sets take effect immediately.

TABLE 3.32 Device parameters in the %disk*n*% parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be FileSystem.
Bus	string	*(Read-only)* The name of the bus on which this disk resides—%Scsi% (or %Scsi*X*%) if a SCSI bus, %Ide% (or %Ide*X*%) if an IDE bus. If the imaging system uses a file system of the native operating system rather than the Adobe storage device implementation, this parameter may not be meaningful or may be absent. More information about the bus (for this disk device) can be obtained by using **currentdevparams** with the value of this parameter as the device of interest.
HasNames	boolean	*(Read-only)* A flag specifying whether the device supports named files. Since a device will not mount successfully unless it contains a valid file system, **HasNames** is always *true* for mounted devices; if *false*, the device is not mounted.
Mounted	boolean	A flag specifying whether the device should be mounted or dismounted, or (if queried) whether the device is currently mounted.

		Setting **Mounted** to *true* indicates that the system should attempt to mount the device, and setting it to *false* that it should attempt 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 unless it contains a valid file system (that is, unless the **HasNames** parameter is *true*). If an attempted mount (or dismount) fails, a **configurationerror** occurs.
		When queried, the value of this parameter indicates whether the device is currently mounted. **Mounted** should be queried immediately after it is set, to obtain the result of the attempted mount or dismount.
Removable	boolean	*(Read-only)* A flag specifying whether the device supports removable media. Depending on how the removable-media device operates, setting the value of **Mounted** to *false* for such a device will either eject the medium or allow the medium to be removed. Once the medium has been removed, the device cannot be mounted again until the medium is reinserted.
Writeable	boolean	*(Read-only except during a mount)* A flag specifying whether the files on the device can be opened for write access (or, if the device is not mounted, whether the device will support writeable media). If the medium is write-protected, **Writeable** is *false*. This parameter can be set only during a mount—that is, at the same time that the **Mounted** parameter is being set to *true*—and only if the medium is not write-protected.
Searchable	boolean	A flag specifying whether the device participates in searches for a file when a file name is supplied that does not specify a device (see *PLR3*, Section 3.8.2).
		Note: *On some products, devices that support removable media will initially have* ***Searchable*** *set to* false. ***Searchable*** *must be explicitly set to* true *on such devices to enable them to participate in device searches.*
SearchOrder	integer	The priority (0 or greater) at which the device participates in file searches as indicated by a value of *true* for the **Searchable** parameter. The lower the value, the higher the priority. This parameter is ignored if **Searchable** is *false*.
BlockSize	integer	*(Read-only)* The number of bytes in a *block* on the formatted device. For a disk using the Adobe file system, **BlockSize** is 1024. This parameter determines the unit for the values of the **Free**, **PhysicalSize**, and **LogicalSize** parameters.
Free	integer	*(Read-only)* The number of blocks of free space available on the medium in the device (where the block size is indicated by the **BlockSize** parameter). **Free** is 0 if the medium is completely full or if the device is not mounted.

PhysicalSize	integer	*(Read-only)* The number of blocks of medium in the device (where the block size is indicated by the **BlockSize** parameter). This is the maximum allowable size of the file system; the exact size is determined by the **LogicalSize** parameter. **PhysicalSize** is 0 if the device is not mounted.
LogicalSize	integer	The number of blocks allocated to the file system (where the block size is indicated by the **BlockSize** parameter). When queried, **LogicalSize** is 0 if the device is not mounted. When set, this parameter specifies the number of blocks to be allocated to the file system when it is created in response to the **InitializeAction** parameter. The value of **LogicalSize** must not exceed the value of **PhysicalSize**. If **LogicalSize** is 0, **InitializeAction** uses a default size that is normally the same as the value of **PhysicalSize**. If **LogicalSize** is set with a certain value and (as specified by the value of **InitializeAction**) the medium in the device is reformatted before the file system is created, a subsequent query of **LogicalSize** should return the value that was set. However, if **LogicalSize** is queried *before* the medium is reformatted, it may return the current size rather than the value that was set for it.
InitializeAction	integer	A code specifying an action for initializing the device: 0 No action. This value is returned when the parameter is queried. 1 Deletes the current file system (if any) and creates one having the size specified by the **LogicalSize** parameter. (The medium is assumed to have been formatted already.) The device must first be mounted; otherwise, an **ioerror** will occur. 2 Reformats the entire medium before creating a new file system of size **LogicalSize**. The **Interleave** parameter also plays a role in how the medium is formatted; see the description of **Interleave** for details. ≥3 Has the same effect as the value 2 and also carries out product-dependent actions, which typically consist of preformatting the disk and running integrity tests before creating the file system. Some devices can have additional parameters that serve as arguments to **InitializeAction**.
Interleave	integer	The interleave number, *n*, specifying *n*-to-1 interleaving. Interleaving arranges logically contiguous sectors on the disk in the most efficient way for the system using that disk. This parameter is used only when the medium is being formatted in response to the **InitializeAction** parameter. For example, assume there are 16 sectors going around a single track on a disk. If the first sector is logical sector number 1, the second 2, the third 3, and so on, the value of **Interleave** is 1 (1-to-1 interleaving). In this case, the

		system must be very fast 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, it has to wait an entire revolution of the disk to get the next sector. This can result in very poor performance. If the first sector is logical sector number 1, the third 2, the fifth 3, and so on, the system will need to be able to consume the current sector while the head skips over a sector in time for the next logical sector. In this case, the value of **Interleave** is 2 (2-to-1 interleaving). The sectors in between are used for higher logical numbers, and it takes a minimum of two revolutions to get the data for an entire track off the disk. In this example, the second *physical* sector on the disk would be between *logical* sectors 1 and 2, and would be logical sector 9. Similarly, with an **Interleave** value of 3, the first sector is logical sector number 1, the fourth 2, and so on. Normally, **Interleave** should be set to a value that allows the software to use the data during the time between sectors, but not waste any time. It is difficult to determine what the proper value is, and the value depends greatly on the job accessing the disk. For drives that provide buffering for a full track of data, 1-to-1 interleaving is almost always most efficient.
PrepareAction	integer	A code specifying an action to prepare the underlying file system for a specific purpose: 0 No action. 1 A product-specific action that loads system files. In one case, these files support older versions of Adobe Japanese typefaces and, on a writeable file system, enable older versions of the Japanese Font Downloader utility to work correctly. If **InitializeAction** and **PrepareAction** are set in the same invocation of **setdevparams**, the actions performed in response to **InitializeAction** precede those performed in response to **PrepareAction**.

TABLE 3.33 Device parameters in the %cartridge% (or %cartridge*n*%) and %rom% (or %rom*n*%) parameter sets

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be FileSystem.
CartridgeType	integer	*(Read-only)* The category classification of the cartridge, from a registry maintained by Adobe.

CartridgeID	integer	*(Read-only)* A number that uniquely identifies this cartridge on a product. This parameter is used by the interpreter to determine whether a cartridge has been removed from a slot and a different cartridge inserted.
HasNames	boolean	*(Read-only)* See Table 3.32 above.
Mounted	boolean	See Table 3.32 above.
Removable	boolean	*(Read-only)* See Table 3.32 above.
Writeable	boolean	*(Read-only except during a mount)* See Table 3.32 above.
Searchable	boolean	See Table 3.32 above.
SearchOrder	integer	See Table 3.32 above.
BlockSize	integer	*(Read-only)* See Table 3.32 above. The value of **BlockSize** for these parameter sets is typically 1.
Free	integer	*(Read-only)* See Table 3.32 above.
PhysicalSize	integer	*(Read-only)* See Table 3.32 above.
LogicalSize	integer	See Table 3.32 above.
InitializeAction	integer	A code specifying an action for initializing the device. The valid values for writeable cartridges are listed below. (Setting **InitializeAction** for a read-only cartridge has no effect.) 0 No action. This value is returned when the parameter is queried. 1 Reformats the entire medium and then creates a new file system using the full size of the cartridge.

TABLE 3.34 Device parameters in the %ram% (or %ram*n*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be FileSystem.
HasNames	boolean	*(Read-only)* A flag specifying whether the device supports named files; always *true*.
Mounted	boolean	*(Read-only)* A flag specifying whether the device is mounted; always *true*.
Removable	boolean	*(Read-only)* A flag specifying whether the device supports removable media; always *false*.
Writeable	boolean	*(Read-only)* A flag specifying whether the files on the device can be opened for write access; always *true*.
Searchable	boolean	*(Read-only)* See Table 3.32 on page 61. This parameter is always *false.*

SearchOrder	integer	*(Read-only)* See Table 3.32 on page 61. This parameter (whose value is always 100) is ignored in this parameter set, since **Searchable** is always *false*.
BlockSize	integer	*(Read-only)* See Table 3.32 on page 61. The value of **BlockSize** for this parameter set is typically 1.
Free	integer	*(Read-only)* The number of blocks of free space available on the medium for the device (where the block size is indicated by the **BlockSize** parameter). **Free** is 0 if the medium is completely full.
PhysicalSize	integer	*(Read-only)* See Table 3.32 on page 61.
LogicalSize	integer	See Table 3.32 on page 61.
InitializeAction	integer	A code specifying an action for initializing the device: 0 No action. This value is returned when the parameter is queried. 1 Reformats the entire medium and then creates a new file system of size **LogicalSize**.

TABLE 3.35 Device parameters in the %os% (or %os*X*%) parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be FileSystem.
HasNames	boolean	*(Read-only)* A flag specifying whether the device supports named files; always *true*.
Mounted	boolean	*(Read-only)* A flag specifying whether the device is mounted; always *true*.
Removable	boolean	*(Read-only)* A flag specifying whether the device supports removable media; always *false*.
Writeable	boolean	*(Read-only)* A flag specifying whether the files on the device can be opened for write access; always *true*.
Searchable	boolean	*(Read-only)* See Table 3.32 on page 61.
SearchOrder	integer	See Table 3.32 on page 61. The value of this parameter is initially 2.
BlockSize	integer	*(Read-only)* The number of bytes in a block (for the partition dedicated to PostScript). This parameter is set by the environment, not by PostScript, and is typically 1024.
Free	integer	*(Read-only)* The number of blocks of free space available in the PostScript partition (where the block size is indicated by the **BlockSize** parameter). **Free** is 0 if the partition is completely full.

PhysicalSize	integer	*(Read-only)* The number of blocks in the PostScript partition (where the block size is indicated by the **BlockSize** parameter).
LogicalSize	integer	*(Read-only)* The number of blocks allocated to the file system (where the block size is indicated by the **BlockSize** parameter). In this parameter set, the values of **LogicalSize** and **PhysicalSize** will always be the same.
InitializeAction	integer	*(Read-only)* A code specifying an action for initializing the device; always 0, indicating no action.

CHAPTER 4

Resources

THIS CHAPTER SUPPLEMENTS Section 3.9 of the *PostScript Language Reference*, Third Edition. It provides information about additional named resources that are relatively device-specific—those that describe features unique to a particular output device or limited to only a few devices.

4.1 Regular Resource Categories

Regular resources are those whose instances are ordinary useful objects, such as font or halftone dictionaries. Most categories of regular resources are described in *PLR3*, but a few of them are product-dependent and are described in this *Supplement* instead. These include the following:

- Instances of **ControlLanguage** are dictionaries that describe the control languages available in a product. A control language is a means for controlling product features, such as default configuration and status reporting.
- Instances of **PDL** are dictionaries that describe the page description language interpreters available in a product. This category supersedes the implicit resource category **Emulator**, since its instances provide a more complete description of each interpreter (or emulator).
- Instances of **Localization** are dictionaries that describe the natural languages (such as English, Japanese, or German) supported by a product—that is, the language of its user interface.
- Instances of **HWOptions** are strings that describe the special hardware options present in a product.

These resource categories are described further in the following sections.

4.1.1 ControlLanguage

The **ControlLanguage** resource category enables an application or printer driver to determine what control languages are available on a given output device. A control language determines how the environment and device parameters are configured, how jobs are identified, and the format of device-generated messages on the back channel. Each instance of the **ControlLanguage** resource category is a dictionary whose contents describe one of the control languages supported by the device. Table 4.1 shows the entries in such a dictionary.

TABLE 4.1 Entries in an instance of the ControlLanguage resource category

KEY	TYPE	VALUE
Selector	name	*(Required)* A name object that can be used as the value of the **PrinterControl** device parameter in any parameter set of type Communications (see Table 3.3 on page 29).
LanguageFamily	string	*(Required)* The name of the control language, such as PostScript or PJL.
LanguageLevel	string	*(Optional)* A string identifying the level of the control language specified by **LanguageFamily** that is implemented on this device. This can be a level identifier, such as 2 (LanguageLevel 2) for the PostScript language family; for a language family such as PJL, which has no such level designations, the value is an empty string.
LanguageVersion	string	*(Optional)* A string identifying the version of the control language. This is typically a microcode version identifier, such as 2015.101; it can also be a product name, such as LaserJet 4si.

4.1.2 PDL

The **PDL** resource category enables an application or printer driver to determine what page description language (PDL) interpreters are available on a given output device. Each instance of this category is a dictionary whose contents describe one of the page description languages supported by the device. Table 4.2 shows entries that are typically present in such a dictionary. Additional entries may be added to further characterize the page description language.

TABLE 4.2 Entries in an instance of the PDL resource category

KEY	TYPE	VALUE
Selector	name	*(Required)* A name object that can be used as the value of the **Interpreter** device parameter in any parameter set of type Communications (see Table 3.3 on page 29). This same name is used as the name of any parameter set associated with the interpreter.
LanguageFamily	string	*(Required)* The name of the page description language, such as PostScript or PCL.
LanguageLevel	string	*(Optional)* A string identifying the level of the page description language specified by **LanguageFamily** that is implemented on this device. This can be a level identifier, such as 2 (LanguageLevel 2) or 5e (PCL 5e), or it can be a product name if no level designation applies.
LanguageVersion	string	*(Optional)* A string identifying the version of the page description language. This is typically a microcode version identifier, such as 2015.101; it can also be a product name, such as LaserJet 4si.

4.1.3 Localization

The **Localization** resource category enables an application or printer driver to determine what natural languages (such as English, Japanese, or German) are available on a given output device. Each instance of this category is a dictionary that identifies a particular language, national dialect, and character set that are supported by the device. Table 4.3 shows the entries in such a dictionary. Note that these are the same as the entries in the %Console% parameter set; see Table 3.28 on page 57 for further information.

TABLE 4.3 Entries in an instance of the Localization resource category

KEY	TYPE	VALUE
Language	name	A name object specifying the language with a 2-character language code from the ISO 639 standard. See Tables 3.28 and 3.29 on pages 57 and 58 for details.
Country	name	A name object that indirectly specifies the dialect of the language by specifying the country in which the dialect is used. The country is indicated by a 2-character country code from the ISO 3166 standard. See Tables 3.28 and 3.30 on pages 57 and 59 for details.

CharSet	name	A name object specifying either a standard character set or a vendor-specific character set. See Table 3.28 on page 57 for details.

Each **Localization** instance represents a particular combination of **Language**, **Country**, and **CharSet** values. In general, any given device will support only a sparse subset of all possible combinations. For example, the entire **Localization** category might consist of the following instances:

```
<<    /Language /EN
      /Country /US
      /CharSet /ISO-646-ISV
>>

<<    /Language /JA
      /Country null
      /CharSet /JIS-...
>>
```

where the code EN denotes English, JA denotes Japanese, US denotes the United States dialect of English, and the null value indicates that no specific dialect is identified for Japanese.

Each **Localization** instance has a unique name. Although no specific naming scheme is enforced, the following guidelines are recommended:

- Instances can be named by concatenating the values of the three dictionary entries, separated by hyphens. For example, the two instances shown above would be named

  ```
  /EN-US-ISO-646-ISV
  ```

  and

  ```
  /JA-JIS-...
  ```

  respectively. This approach yields a name indicative of the dictionary's contents.

- An alternative is to provide descriptive names for the instances. For example, the two instances shown above might be named

  ```
  /AmericanEnglish
  ```

  and

```
/Japanese
```

Either naming scheme provides a simple way for a PostScript application or printer driver to specify the language to be used with the device. Using the second naming scheme, for example, the following PostScript code changes the language to Japanese:

```
% Set the console to use the Japanese language
   (%Console%)
   /Japanese  /Localization  findresource
setdevparams
```

Note: *If the system parameter* **SystemParamsPassword** *is set, it is also necessary to include a valid* **Password** *entry in the* **Localization** *resource or run this code as an unencapsulated system administrator job; for details, see* PLR3*, Section C.3.1.*

4.1.4 HWOptions

Instances of the **HWOptions** resource category are strings identifying any special hardware options that may be present on a given output device. This category is available if needed to address the special requirements of a particular device; however, because the exact details are device-dependent, no specific instances of the category are officially defined at the time of publication.

4.2 Implicit Resource Categories

Implicit resources represent some built-in capability of the PostScript interpreter. For example, instances of the **Filter** category are filter names, such as **ASCII85Decode** and **CCITTFaxDecode**, that are passed directly to the **filter** operator. Table 4.4 lists implicit resources that are not described in *PLR3*.

TABLE 4.4 Implicit resources

CATEGORY NAME	INSTANCES		
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%**
	%TokenTalk%	**%TokenTalk_NV%**	**%TokenTalk_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%	%LAT_NV%	%LAT_Pending%
	%SNMP%	%SysLog%	%TCP%
	%UDP%	%IP%	%SPX%
	%IPX%	%EthernetPhysical%	%TokenRingPhysical%
	%Scsi%	%Ide%	%Engine%
	%Console%	%Calendar%	
	%disk*n*%	%cartridge%	%rom%
	%ram%	%os%	
	%PCL%	%LaserJetIII%	%LaserJetIIP%
	%Diablo630%	%HP7475A%	
Emulator	LaserJetIII	LaserJetIIP	HP7475A
	Diablo630	EpsonFX850	ProprinterXL
	PCL		
Filter	GIFDecode	PNGDecode	

Instances of the **IODevice** category refer to device parameter sets described in Chapter 3. The **Emulator** resource category has been superseded by the **PDL** resource in LanguageLevel 3 and may not be present on all products.

GIFDecode and **PNGDecode** are optional filters that are used only as part of the implementation of Web printing (printing HTML content from the World Wide Web). PostScript programs should not invoke these filters.

CHAPTER 5

Other Extensions

THIS CHAPTER PROVIDES information about miscellaneous extensions to the PostScript language that are not described in the *PostScript Language Reference*, Third Edition. These include new and extended font types and extensions to the contents of image dictionaries and trapping parameter dictionaries. The chapter also provides guidelines for synchronizing PostScript color rendering dictionaries (CRDs) with device color profiles as defined by the International Color Consortium (ICC).

5.1 Font Extensions

This section supplements Chapter 5 of the *PostScript Language Reference*, Third Edition. It describes:

- The %fontset% fictitious file system, the compatibility mechanism that ensures that all programs will work with fonts defined in **FontSet** resource instances
- A new kind of Type 0 CIDFont: a CFF CIDFont, defined in **FontSet** resources rather than in a CIDFont file
- Extensions to the Type 2 CIDFont dictionary to accommodate incremental downloading of CJK TrueType fonts

5.1.1 %fontset% Fictitious File System

As described in *PLR3*, Section 5.8.1, Type 2 (Compact Font Format, or CFF) and Type 14 (Chameleon) fonts are integrated into the PostScript font model in a way that is compatible with both LanguageLevel 1 (file system) and LanguageLevel 2 (resource) methods for accessing fonts. There exists a fictitious file system named

%fontset%, whose “files” are the constituent fonts in all available **FontSet** resource instances. This file system has the following characteristics and behavior:

- It is read-only.
- Its “files” appears to have names of the form fonts/Palatino. The prefix is the constant string fonts/, which is the recommended value of the **FontResourceDir** system parameter; this is followed by the name of a font in some **FontSet** resource instance.
- It is typically searchable—that is, the default value of its **Searchable** parameter is usually *true*. This means that it is among the file systems considered when a file name is supplied that does not specify a device (see *PLR3*, Section 3.8.2).
- Font lookup operations will tend to cause all **FontSet** resource instances to be loaded into virtual memory (VM) as a side effect. A **FontSet** dictionary takes minimal VM (a few hundred bytes) if loaded from a positionable file. Furthermore, **FontSet** instances are expected to be few in number. (Most printers will have at most three—one for all CFF fonts and one each for a Chameleon master font and its descriptor database.)
- File systems are searched in order of their **SearchOrder** device parameter. Thus, the %fontset% file system will be consulted before or after other file systems, such as cartridges or disks, according to their relative **SearchOrder** values. Specifying that %fontset% be searched first will yield the best performance for fonts present in the **FontSet** dictionaries (since no other file systems need to be consulted), but it will preclude overriding a **FontSet** font by installing a font with the same name on disk.
- The **Searchable** and **SearchOrder** parameters of the %fontset% file system can be changed, with the **setdevparams** operator.

Opening a file with

```
(%fontset%fonts/Palatino) (r) file
```

or (if %fontset% is the file system selected by the search rule) with

```
(fonts/Palatino) (r) file
```

causes the **FontSet** resource instances to be enumerated until one containing the named font is found. The resource instances are loaded into VM as a side effect. A fictitious file is then fabricated and made available for reading. When this fictitious file is executed, it causes the font dictionary to be built and defined as an instance of the **Font** resource category. The contents of the file are undocumented and subject to change.

***Note:** The fictitious file does not have any document structuring conventions (DSC); consequently, a **Font resourcestatus** operation will not find a %%VMusage: comment and will return a size of –1.*

The resulting font dictionary has a **FontType** value of 2 for CFF or 14 for Chameleon, plus all the required entries for a Type 1 font dictionary (as defined in *PLR3*, Tables 5.2, 5.3, and 5.4 on pp. 324–327), except that it does not have a **Private** entry. It may also contain additional entries that are undocumented and subject to change.

A PostScript program can copy the font dictionary and insert or modify entries as specified in the aforementioned tables. These modifications have the same effects for Type 2 and Type 14 fonts as for Type 1 fonts.

Executing the **status** operator for a valid file name in the %fontset% file system yields four meaningless numbers followed by *true* if the named font exists in any font set; otherwise it yields *false*.

5.1.2 CFF CIDFonts

Type 2 (Compact Font Format, or CFF) base fonts and Type 0 CIDFonts are both based on Type 1 font technology. There is also a special kind of Type 0 CIDFont: a CFF CIDFont. Other Type 0 CIDFonts are defined in CIDFont files, but CFF CIDFonts, like Type 2 base fonts, are defined in **FontSet** resources: they are integrated into the PostScript font model via the %fontset% fictitious file system, as described above.

Like other Type 0 CIDFonts, a CFF CIDFont is characterized by **CIDFontType** and **FontType** values of 0 and 9, respectively. It is further characterized by a **FontType** value of 2 in each **FDArray** subdictionary.

For more information, see Section 5.8.1 (pp. 343–346) and "Type 0 CIDFonts" (pp. 371–376) in *PLR3*.

5.1.3 Incremental Downloading of CJK TrueType CIDFonts

As described in *PLR3*, Section 5.11, CID-keyed fonts provide great flexibility for representing text in writing systems for languages with large character sets, such as Chinese, Japanese, and Korean (CJK). CJK TrueType fonts can be downloaded as CID-keyed fonts containing Type 2 CIDFonts, which are the analog of Type 42 (TrueType) base fonts.

Applications or drivers usually download CJK fonts as CIDFonts into PostScript VM or install them on a printer disk. Some drivers, however, particularly Windows® drivers, need to be able to download fonts incrementally or as subsets to minimize the transmission cost of a print job. To accommodate incremental downloading of CJK TrueType fonts, entries in the Type 2 CIDFont dictionary have been extended as follows:

- The definition of the **CIDMap** entry can be a dictionary or an integer as well as a string or an array, as described in the next section.
- The definition of the **GlyphDirectory** entry has been extended, and an optional **MetricsCount** entry added, to allow the inclusion of glyph metric information in **GlyphDirectory**, as described in the section "GlyphDirectory and MetricsCount" on page 79.

New CIDMap Types

The value of **CIDMap** can be not only a string or an array of strings but also a dictionary or an integer.

CIDMap can be defined as a dictionary that maps CIDs directly to glyph indices, as follows:

```
/CIDMap
    <<    CID0 GI0
          …
          CIDn GIn
    >>
```

where $CID_0 \ldots CID_n$ are the CIDs—that is, 0 through (**CIDCount** − 1)—and $GI_0 \ldots GI_n$ are the corresponding glyph indices. The glyph indices are then used to locate the corresponding glyph descriptions in **GlyphDirectory**. The **CIDMap** dictionary can be downloaded incrementally as glyph descriptions are needed.

If a driver can map CIDs to glyph indices directly, it is no longer necessary to download **CIDMap** at all; this is where defining **CIDMap** as an integer comes in. The mapping relationship can be defined so that there is a constant integer offset from the identity mapping, as follows:

```
/CIDMap integer def
```

With this definition, each CID maps to a glyph index that is the sum of the CID and *integer*. This allows fonts having many glyphs to be represented as several separate CIDFonts. A typical Korean font, for example, can contain 22,000 glyphs, which would be accessed using glyph indices 0 through 21,999. Because of PostScript limitations, such a font would have to be represented as two separate CIDFonts, for which *integer* would be 0 and 16,000, respectively. The **CIDMap** for the first CIDFont would be defined as

```
/CIDMap 0 def
```

so that the CID 0 maps to the glyph index 0, the CID 1 to the glyph index 1, and so on, up to 15,999. The **CIDMap** for the second CIDFont would be defined as

```
/CIDMap 16000 def
```

so that the CID 0 maps to the glyph index 16,000, the CID 1 to the glyph index 16,001, and so on, until the CID 5999 maps to 21,999.

GlyphDirectory and MetricsCount

To facilitate incremental downloading of Type 2 CIDFonts, glyph metrics may optionally be defined in the **GlyphDirectory** entry in the Type 2 CIDFont dictionary (as an alternative to defining them in the TrueType "hmtx" and "vmtx" tables embedded in the **sfnts** array). Including glyph metric information in **GlyphDirectory** for a subset of characters avoids the need to download large tables, such as the "hmtx" and "vmtx" tables. Whether the metrics are included in **GlyphDirectory**, and which writing mode or modes they apply to, is defined in the **MetricsCount** entry in the Type 2 CIDFont dictionary.

The optional **MetricsCount** entry—an integer with a default value of 0—specifies the location of the glyph metric information. **MetricsCount** may have a value of 0, 2, or 4, indicating that the first 0, 4, or 8 bytes of the glyph description contain the metric information—that is, the glyph description begins **MetricsCount** × 2 bytes from the beginning of the string.

As described in *PLR3*, Section 5.4, the glyph metric information includes:

- The glyph's *width* (the distance the current point moves when the glyph is shown, not the dimensions of the glyph outline)
- The glyph's *left sidebearing*

These are the general terms used for these metrics, independent of the writing mode (that is, whether **WMode** is 0 for horizontal or 1 for vertical); however, for writing mode 1, the metrics are actually the glyph's height and top sidebearing.

If **MetricsCount** is 0:

- Glyph metric information for writing mode 0 is obtained from the "hmtx" table or the **Metrics** dictionary.
- Glyph metric information for writing mode 1 is obtained from the "vmtx" table, the **Metrics2** dictionary, or **CDevProc**.

If **MetricsCount** is 2:

- Glyph metric information for writing mode 0 is obtained from the width and left sidebearing data at the beginning of the glyph description defined in **GlyphDirectory**.
- Glyph metric information for writing mode 1 is obtained from the "vmtx" table, the **Metrics2** dictionary, or **CDevProc**.

If **MetricsCount** is 4:

- Glyph metric information for writing mode 0 is obtained from bytes 4 and 5 (for the width) and bytes 6 and 7 (for the left sidebearing) at the beginning of the glyph description defined in **GlyphDirectory**.
- Glyph metric information for writing mode 1 is obtained from bytes 0 and 1 (for the height) and bytes 2 and 3 (for the top sidebearing) at the beginning of the glyph description defined in **GlyphDirectory**.

Table 5.1 summarizes the meanings of the different values of **MetricsCount**.

TABLE 5.1 Meanings of MetricsCount values

MetricsCount VALUE	BYTES AT BEGINNING OF GLYPH DESCRIPTION	SOURCE OF METRICS FOR WMode 0 (HORIZONTAL)	SOURCE OF METRICS FOR WMode 1 (VERTICAL)
0	0	"hmtx" table or **Metrics**	"vmtx" table, **Metrics2**, or **CDevProc**
2	4	Bytes at beginning of glyph description: Bytes 0, 1: Width Bytes 2, 3: Left sidebearing	"vmtx" table, **Metrics2**, or **CDevProc**
4	8	Bytes at beginning of glyph description: Bytes 4, 5: Width Bytes 6, 7: Left sidebearing	Bytes at beginning of glyph description: Bytes 0, 1: Height Bytes 2, 3: Top sidebearing

5.2 Interpolation Details Dictionary

A new optional entry (see Table 5.2) has been added to all image dictionaries, regardless of type. Image dictionaries are described in *PLR3*, Section 4.10.5.

TABLE 5.2 Additional entry in all image dictionaries

KEY	TYPE	SEMANTICS
InterpolationDetails	dictionary	*(Optional)* A dictionary containing device-specific information related to image interpolation. Like all details dictionaries, this dictionary should include a **Type** entry whose integer value determines the remaining structure and contents of the dictionary (see Section 2.1, "Details Dictionaries," as well as *PLR3*, Section 6.1.2). This parameter is ignored if the value of **Interpolate** is *false*.

5.3 Imagemask Trapping Parameter

A new optional trapping parameter (see Table 5.3) has been added to the trapping parameter dictionary. Trapping parameter dictionaries are described in *PLR3*, Section 6.3.3.

TABLE 5.3 Additional trapping parameter

KEY	TYPE	SEMANTICS
ImagemaskTrapping	boolean	*(Optional)* A flag specifying whether trapping should be performed between stencil masks and the background against which they are painted. When this flag is *false*, stencil masks are painted within the given trapping zone with an implicit colorant type of OpaqueIgnore (see **ColorantType** in *PLR3*, p. 444). That is, no traps are generated between the stencil mask and other objects that overlay it or that it overlays. Note, however, that objects that overlay the stencil mask and overlap each other *will* be trapped to each other. Default value: *true*.

When a page description contains a stencil mask to be drawn with the **imagemask** operator (*PLR3*, pp. 302–303 and 608–610), it frequently is not appropriate to trap the stencil mask against the background on which it is painted. Particularly when two or more stencil masks overlap or when the area covered by a stencil mask is small, trapping can produce undesirable smear effects. The trapping parameter **ImagemaskTrapping** can be used to enable or disable such trapping.

5.4 Color Rendering Dictionaries and ICC Profiles

The International Color Consortium (ICC) has defined a standard format, known as an *ICC profile*, for describing the color properties of an output device. ICC profiles can contain the same information as PostScript color rendering dictionaries (CRDs), and can be translated to or from CRDs in a straightforward way. Color rendering dictionaries are described in *PLR3*, Section 7.1. This section describes how to coordinate CRDs on the output device with ICC profiles on the host computer, in order to reduce the memory requirements and file transmission times for color transformations. Such equivalent CRDs and profiles on the device and host are known as *companions*.

Every CRD should include a **CreationDate** entry (*PLR3*, p. 468), whose value is a string indicating the date and time at which the CRD was created or most recently modified. This value should be coordinated with the **calibrationDateTimeTag** entry of any companion profile. (Even if no companion profile is present on the host, the **CreationDate** entry should still be included in the CRD.) The format of the **CreationDate** string should match that of the **calibrationDateTimeTag** entry in the ICC profile, as defined in the document *ICC Profile Format Specification* (see the Bibliography):

YYYYMMDDHHmmSSOHH′mm′

where

YYYY is the year
MM is the month
DD is the day (01–31)
HH is the hour (00–23)
mm is the minute (00–59)
SS is the second (00–59)
O is the relationship of local time to Universal Time (UT); see below
HH′ is the absolute value of the offset from UT in hours (00–23)
mm′ is the absolute value of the offset from UT in minutes (00–59)

Fields after the year are optional. The default values for *MM* and *DD* are both 01; all other numerical fields default to zero values. A plus sign (+) as the value of the *O* field indicates that local time is later than UT, a minus sign (-) indicates that local time is earlier than UT, and the letter Z indicates that local time is equal to UT. If no UT information is specified, the relationship of the specified time to UT 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 entry **crdInfoTag**, containing the PostScript product name to which this profile corresponds and the names of the companion CRDs. Note that a single profile can have multiple companion CRDs, corresponding to different rendering intents (see *PLR3*, Section 7.1.3). Table 5.4 gives the format of the **crdInfoTag** entry.

TABLE 5.4 Format of crdInfoTag

BYTES	CONTENT
0–3	Type descriptor crdi (0x63726469)
4–7	Reserved, must be zero
8–11	Character count m of PostScript product name, including terminating null character
12–$(M-1)$	PostScript product name string in 7-bit ASCII (where $M = 12 + m$)

M–$(M+3)$	Character count n of CRD name for rendering intent 0, including terminating null character
$(M+4)$–$(N-1)$	CRD name string for rendering intent 0, in 7-bit ASCII (where $N = M + n + 4$)
N–$(N+3)$	Character count p of CRD name for rendering intent 1, including terminating null character
$(N+4)$–$(P-1)$	CRD name string for rendering intent 1, in 7-bit ASCII (where $P = N + p + 4$)
P–$(P+3)$	Character count q of CRD name for rendering intent 2, including terminating null character
$(P+4)$–$(Q-1)$	CRD name string for rendering intent 2, in 7-bit ASCII (where $Q = P + q + 4$)
Q–$(Q+3)$	Character count r of CRD name for rendering intent 3, including terminating null character
$(Q+4)$–$(R-1)$	CRD name string for rendering intent 3, in 7-bit ASCII (where $R = Q + r + 4$)

CRDs are stored on the output device as instances of resource category **Color-Rendering** with the names specified in the profile's **crdInfoTag** entry. If no companion CRD exists for a given rendering intent, the corresponding character count is zero and there is no CRD name string.

The CRD's **CreationDate** entry and the ICC profile's **crdInfoTag** entry can be coordinated differently depending on whether bidirectional communications are available between the host and the output device and whether the CRD was supplied with the output device or was downloaded from a host in the field. Bidirectional communication allows the output device to be queried to determine the availability of a given CRD and its associated **CreationDate** entry.

In the absence of bidirectional communications, the list of device-resident CRDs and their **CreationDate** entries is available through the output device's PostScript printer description (PPD) and (on some platforms) the host profile registry. PPDs currently contain the names of the CRDs that ship with an output device; the PPD format will be extended to contain the **CreationDate** entry for each CRD as well. The registry should be updated whenever CRDs are downloaded or created in the field.

To determine whether a suitable CRD is available on the output device, a PostScript driver can do the following:

1. Obtain the device's PostScript product name, either from the PPD or via the PostScript **product** operator, and compare it with the product names given in the **crdInfoTag** entries of all available ICC profiles on the host. This comparison limits the selection of profiles to only those pertaining to the given output device.
2. For each profile whose product name matches that of the device, look in the **crdInfoTag** entry for the CRD name corresponding to the required rendering intent and check whether the device has a **ColorRendering** resource with that name.
3. If a resource with the specified name exists, compare its **CreationDate** entry with the profile's **calibrationDateTimeTag** entry. If the two match, the resource contains the required CRD.

If no suitable CRD is found on the device, the driver can download one from an ICC profile on the host. Ordinarily, such a downloaded CRD will not be available for subsequent jobs, but the driver can make it persistent by giving it an appropriate **CreationDate** entry, updating the registry, and updating the profile's **crdInfoTag** entry to refer to this CRD.

If a CRD on the output device has no corresponding profile on the host, the driver can coordinate it with a companion profile either by copying its **CreationDate** entry to the profile or by updating the CRD and registry from the profile's **calibrationDateTimeTag** entry. In either case, the profile's **crdInfoTag** entry must be filled in correctly. Note also that in this case the CRD updates may not persist across power cycles of the output device; such power cycles should result in the registry being updated.

CHAPTER 6

Compatibility

MOST POSTSCRIPT IMPLEMENTATIONS provide a standard dictionary named **statusdict** to hold product-specific operators and other data whose names and values vary from one product to another (and sometimes even from version to version of the same product). Information in **statusdict** is associated with unique features of a product that cannot be accessed in any standard way. Although the contents of **statusdict** are product-dependent, an attempt has been made to maintain a consistent specification for common features that are found in more than one product.

In LanguageLevel 1, **statusdict** includes a variety of operators to select print-engine features, set communication parameters, and control other product-specific aspects of the PostScript interpreter's operating environment. In LanguageLevels 2 and 3, most of these functions have been subsumed by standard operators, such as **setsystemparams**, **setdevparams**, and **setpagedevice**. However, LanguageLevel 2 and LanguageLevel 3 implementations continue to provide a set of *compatibility operators* and related data values corresponding to the older features, in order to maintain compatibility with existing LanguageLevel 1 software that might depend on their presence. These compatibility operators and data values are the subject of this chapter.

Unlike the standard PostScript operators described in the *PostScript Language Reference*, Third Edition (which reside in the standard dictionary **systemdict**), the majority of compatibility operators and data values are defined in **statusdict** (though a few of them are in **systemdict** or **userdict** instead). Because the availability of specific operators and data values is product-dependent, their continued use beyond LanguageLevel 1 is not recommended. The appropriate LanguageLevel 2 and LanguageLevel 3 operators should be used instead.

Note: *The behavior of many of the compatibility operators is defined in terms of setting or reading corresponding system, user, device, or page device parameters. A given compatibility operator typically will be defined only if its associated parameter exists; however, this is not invariably the case.*

Note also that some of the compatibility "operators" are actually implemented as PostScript procedures rather than true operators. These procedures are designed to behave exactly the same as the original LanguageLevel 1 operators; for example, they mimic the exact error behavior of the original operators, even where it differs from that of the more general LanguageLevel 2 and LanguageLevel 3 operators that have replaced them. The distinction between procedures and true operators is thus immaterial from the viewpoint of the calling program.

This chapter is divided into two sections:

- Section 6.1 gives a summary of the operators and data values, organized into groups of related functions. The summary is intended to help locate the operators needed to perform specific tasks.
- Section 6.2 gives detailed descriptions of all operators and data values, organized alphabetically by name.

6.1 Compatibility Operator Summary

System Configuration

string	**setprintername** –	Set device name
string	**printername** *substring*	Return device name
–	**product** *string*	Value of product name string
–	**revision** *int*	Value of product revision level
–	**buildtime** *int*	Value of interpreter time stamp
–	**byteorder** *bool*	Value of native byte order
–	**realformat** *string*	Value of native real number format
–	**ramsize** *int*	Return available RAM size
–	**pagecount** *int*	Return device page count
bool	**setdosysstart** –	Set system startup mode
–	**dosysstart** *bool*	Return system startup mode
bool	**setdostartpage** –	Set start page flag
–	**dostartpage** *bool*	Return start page flag
bool	**setdoprinterrors** –	Set error printing mode
–	**doprinterrors** *bool*	Return error printing mode

Job Control

–	**jobname**	*string*	Value of job name string
string	**checkpassword**	*bool*	Check password string for **startjob** or **exitserver**
int	**checkpassword**	*bool*	Check password integer for **startjob** or **exitserver**
job manualfeed wait	**setdefaulttimeouts**	–	Set default job, manual feed, and wait timeout intervals
–	**defaulttimeouts**	*job manualfeed wait*	Return default job, manual feed, and wait timeout intervals
int	**setjobtimeout**	–	Set job timeout interval
–	**jobtimeout**	*int*	Return job timeout interval
–	**waittimeout**	*int*	Value of wait timeout interval

Device Control

pixelsperinch	**setresolution**	–	Set pixel resolution
–	**resolution**	*pixelsperinch*	Return horizontal pixel resolution
–	**processcolors**	*int*	Return number of color components
top left	**setmargins**	–	Set page margins
–	**margins**	*top left*	Return page margins
bool	**setmirrorprint**	–	Set mirror printing flag
–	**mirrorprint**	*bool*	Return mirror printing flag
bool	**setpagestackorder**	–	Set page stacking order to complement
–	**pagestackorder**	*bool*	Return complement of page stacking order
–	**manualfeed**	*bool*	Manual feed flag
–	**manualfeedtimeout**	*int*	Manual feed timeout interval

Media Selection

–	**letter**	–	Select letter-size paper
–	**lettersmall**	–	Select letter-size paper with smaller printable area
–	**lettertray**	–	Select letter-size paper tray
–	**legal**	–	Select legal-size paper
–	**legaltray**	–	Select legal-size paper tray
–	**ledger**	–	Select ledger-size paper
–	**ledgertray**	–	Select ledger-size paper tray

–	**11x17**	–	Select 11-by-17-inch paper
–	**11x17tray**	–	Select 11-by-17-inch paper tray
–	**a3**	–	Select A3 paper
–	**a3tray**	–	Select A3 paper tray
–	**a4**	–	Select A4 paper
–	**a4small**	–	Select A4 paper with smaller printable area
–	**a4tray**	–	Select A4 paper tray
–	**b5**	–	Select B5 paper
–	**b5tray**	–	Select B5 paper tray
–	**note**	–	Select current paper size with smaller printable area

Page Duplexing

bool	**setduplexmode**	–	Set duplex printing flag
–	**duplexmode**	*bool*	Return duplex printing flag
–	**firstside**	*bool*	Test whether current page prints on front side of output medium
–	**newsheet**	–	Print current page if back side
bool	**settumble**	–	Set tumble flag
–	**tumble**	*bool*	Return tumble flag

Imagesetter Compatibility

bool	**setaccuratescreens**	–	Set accurate screens flag
–	**accuratescreens**	*bool*	Return accurate screens flag
freq angle	**checkscreen**	*actualfreq actualangle moirélength*	Return actual screen frequency and angle
width height orientation	**setpage**	–	Set page dimensions and orientation
left	**setpagemargin**	–	Set page offset
–	**pagemargin**	*left*	Return horizontal page offset
width height offset orientation	**setpageparams**	–	Set page dimensions, offset, and orientation
–	**pageparams**	*width height offset orientation*	Return page dimensions, offset, and orientation

Storage Device Compatibility

string	**devmount**	*bool*	Mount storage device
string	**devdismount**	–	Dismount storage device
string blocks action	**devformat**	–	Initialize storage device
string	**devstatus**	*searchable writeable hasNames mounted removable searchOrder free logicalSize true*	
or		*false*	Return storage device attributes
proc scratch	**devforall**	–	Enumerate storage devices

Disk Compatibility

–	**diskonline**	*bool*	Test whether writeable disk mounted
–	**diskstatus**	*free total*	Return available disk space
blocks action	**initializedisk**	–	Initialize all writeable disks
int	**setuserdiskpercent**	–	Formerly set percentage of disk reserved for user files
–	**userdiskpercent**	*int*	Formerly returned percentage of disk reserved for user files

Channel Compatibility

int	**sethardwareiomode**	–	Open channel for communication
–	**hardwareiomode**	*int*	Return currently enabled communication channel
int	**setsoftwareiomode**	–	Set interpreter and protocol
–	**softwareiomode**	*int*	Return interpreter and protocol
stream emulator	**emulate**	–	Yield control to emulator
stream dict emulator	**emulate**	–	Yield control to emulator with parameters
–	**appletalktype**	*string*	Value of AppleTalk type string
channel	**sccbatch**	*baud options*	Return serial communication parameters
channel	**sccinteractive**	*baud options*	Formerly returned serial communication parameters
channel baud options	**setsccbatch**	–	Set serial communication parameters
channel baud options	**setsccinteractive**	–	Formerly set serial communication parameters

6.2 Compatibility Operator Details

a3 – **a3** – (***userdict*** *dictionary)*

requests A3-size paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value of 297 by 420 millimeters ([842 1191] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing A3 as the page size whether or not an A3 media source is available on the device (see *PLR3*, pp. 434–435).

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **a3tray**

a3tray – **a3tray** – (***statusdict*** *dictionary)*

requests a paper tray containing A3-size paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value of 297 by 420 millimeters ([842 1191] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if an A3 media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **a3tray** operator converts any such **configurationerror** to a **rangecheck** error.

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck**
See also: **a3**

a4 – **a4** – *(**userdict** dictionary)*

requests A4-size paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value of 210 by 297 millimeters ([595 842] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing A4 as the page size whether or not an A4 media source is available on the device (see *PLR3*, pp. 434–435).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **a4small, a4tray**

a4small – **a4small** – *(**userdict** dictionary)*

requests A4-size paper as the output medium, with a margin of approximately 1⁄3 inch at all four edges of the physical medium. It invokes the **setpagedevice** operator with a **PageSize** value of 210 by 297 millimeters ([595 842] in default user space units) and an **ImagingBBox** value of [25 25 570 817] (inset 25 units from each edge of the physical sheet).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing A4 as the page size whether or not an A4 media source is available on the device (see *PLR3*, pp. 434–435).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **a4small, a4tray**

a4tray – **a4tray** – *(**statusdict** dictionary)*

requests a paper tray containing A4-size paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value of 210 by 297 millimeters ([595 842] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if an A4 media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **a4tray** operator converts any such **configurationerror** to a **rangecheck** error.

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck**
See also: **a4, a4small**

accuratescreens – **accuratescreens** *bool* *(**statusdict** dictionary)*

returns the current value of the user parameter **AccurateScreens** (*PLR3*, pp. 749 and 757). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setaccuratescreens**

appletalktype – **appletalktype** *string* *(**statusdict** dictionary)*

is a string with the same value as the **LocalTalkType** device parameter in the %LocalTalk% parameter set and the **EtherTalkType** device parameter in the %EtherTalk% parameter set (see Tables 3.12 on page 42 and 3.13 on page 43).

appletalktype is not an operator; it is a name in **statusdict** associated with the AppleTalk type string. LanguageLevel 1 applications that formerly specified the AppleTalk type by setting the value of this string, using code such as

```
/appletalktype (SpoolerControlledPrinter) def
```

can continue to do so; the PostScript interpreter will automatically update the values of both the **LocalTalkType** and **EtherTalkType** device parameters to match.

Similarly, changes in the value of either the **LocalTalkType** or **EtherTalkType** device parameter will affect the value of the **appletalktype** string.

Errors: stackoverflow, undefined

b5 – **b5** – (***userdict*** *dictionary)*

requests B5-size paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value representing this size and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for the specified page size on the current output device (see *PLR3*, p. 414).

Note: *Two different definitions are in common use for the B5 paper size (see Table 3.27 on page 56). Most PostScript implementations will request a* ***PageSize*** *value of [499 709] in default user space units (72nds of an inch), corresponding to the International Organization for Standardization (ISO) definition of 176 × 250 mm; implementations intended for the Japanese market will typically set* ***PageSize*** *to [516 729], representing the Japanese Industrial Standards Committee (JIS) definition of 182 × 257 mm.*

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing B5 as the page size whether or not a B5 media source is available on the device (see *PLR3*, pp. 434–435).

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: limitcheck
See also: b5tray

b5tray – **b5tray** – (***statusdict*** *dictionary)*

requests a paper tray containing B5-size paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value representing this size and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for the specified page size on the current output device (see *PLR3*, p. 414).

Note: *Two different definitions are in common use for the B5 paper size (see Table 3.27 on page 56). Most PostScript implementations will request a* ***PageSize*** *value of [499 709] in default user space units (72nds of an inch), corresponding to the*

International Organization for Standardization (ISO) definition of 176 × 250 mm; implementations intended for the Japanese market will typically set ***PageSize*** *to [516 729], representing the Japanese Industrial Standards Committee (JIS) definition of 182 × 257 mm.*

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if a B5 media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **b5tray** operator converts any such **configurationerror** to a **rangecheck** error.

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3*, pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: limitcheck, rangecheck
See also: b5

buildtime

– **buildtime** *int* *(**statusdict** dictionary)*

is a read-only integer with the same value as the system parameter **BuildTime** (*PLR3*, p. 751). **buildtime** is not an operator; it is a name in **statusdict** associated with the integer value.

Errors: stackoverflow

byteorder

– **byteorder** *bool* *(**statusdict** dictionary)*

is a read-only boolean value with the same value as the system parameter **ByteOrder** (*PLR3*, p. 751). **byteorder** is not an operator; it is a name in **statusdict** associated with the boolean value.

Errors: stackoverflow

checkpassword

string **checkpassword** *bool* *(**statusdict** dictionary)*
int **checkpassword** *bool*

checks whether the operand is a valid password for the **startjob** or **exitserver** operator. **checkpassword** returns *true* if the operand is equal to either the **SystemParamsPassword** or **StartJobPassword** system parameter; see *PLR3*, Section C.3.1

(pp. 754–755). (If an integer operand is supplied, it is converted to a string for purposes of password comparison.) If either **SystemParamsPassword** or **StartJob-Password** is not set (that is, is the empty string), **checkpassword** returns *true*.

There is no LanguageLevel 2 equivalent for this operator.

Errors: **stackunderflow, typecheck**

checkscreen *freq angle* **checkscreen** *actualfreq actualangle moirélength* (***statusdict*** *dictionary)*

returns the actual screen frequency and angle that would result if the **setscreen** operator were called with the operand values *freq* and *angle*. The values returned for *actualfreq* and *actualangle* are the same as for the **ActualFrequency** and **ActualAngle** entries in a type 1 halftone dictionary (*PLR3*, pp. 487–488). *moirélength* is the distance in inches at which the deviation from the requested dot pattern would reach a fixed fraction of the cell size, and is thus a measure of how accurately the actual screen would approximate the requested screen. Note that this operator does not affect the current screen.

There is no LanguageLevel 2 equivalent for this operator.

Errors: **stackoverflow, stackunderflow, typecheck**

defaulttimeouts – **defaulttimeouts** *job manualfeed wait* (***statusdict*** *dictionary)*

returns the current values of the system parameters **JobTimeout** and **WaitTimeout** (see Table 3.2 on page 15) and the page device parameter **ManualFeedTimeout** (Table 2.2 on page 5) as *job*, *wait*, and *manualfeed*, respectively. This operator always succeeds and always returns three result values; if any of the required parameters is not present, the corresponding result value is 0.

Errors: **stackoverflow**
See also: **setdefaulttimeouts**

devdismount *string* **devdismount** –

attempts to dismount the storage device identified by *string* by setting the device parameter **Mounted** (see Table 3.32 on page 61) to *false* in the parameter set corresponding to the device. If the device is not currently mounted, this operator has no effect. Some devices cannot be dismounted; attempting to dismount such a device will also have no effect.

This operator should be invoked only from within a system administrator job.

Errors: **invalidaccess, stackunderflow, typecheck, undefinedfilename**

devforall *proc scratch* **devforall** –

enumerates all known storage devices. For each device, **devforall** copies the device's name into the supplied *scratch* string, pushes a string object designating the substring of *scratch* 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. The value of the* ***HasNames*** *parameter for all such devices will be* false *(see Table 3.3 on page 29).*

There is no LanguageLevel 2 equivalent for this operator (although its effect can be approximated by applying **resourceforall** to the **IODevice** resource category).

Errors: **invalidaccess, rangecheck, stackoverflow, stackunderflow, typecheck, undefined**

devformat *string blocks action* **devformat** –

initializes the storage device identified by *string* by setting the device parameters **LogicalSize** and **InitializeAction** (see Tables 3.32–3.35 on pages 61–66) to the values *blocks* and *action* + 1, respectively, in the parameter set corresponding to the device.

This operator should be invoked only from within a system administrator job.

Errors: **invalidaccess, limitcheck, rangecheck, stackunderflow, typecheck, undefined, undefinedfilename**

devmount *string* **devmount** *bool*

attempts to mount the storage device identified by *string* by setting the device parameter **Mounted** (see Table 3.32 on page 61) to *true* in the parameter set corresponding to the device. The boolean result *bool* is set to *true* if the device is successfully mounted (or was already mounted), or to *false* if the device cannot be mounted at this time.

This operator should be invoked only from within a system administrator job.

Errors: **invalidaccess, stackunderflow, undefinedfilename**

devstatus

string **devstatus** *searchable writeable hasNames mounted*
removable searchOrder free logicalSize true *(if device known)*
false *(if not)*

returns the values of various file system attributes for the disk device identified by *string* (which must be %disk*n*% for some integer value of *n*). If the specified device name is known, the boolean value *true* is returned on the top of the stack, accompanied by the values of the device parameters **Searchable**, **Writeable**, **HasNames**, **Mounted**, **Removable**, **SearchOrder**, **Free**, and **LogicalSize** (see Table 3.32 on page 61). If the device name is unknown, only the value *false* is returned on the stack.

Errors: **stackoverflow, stackunderflow, typecheck**

diskonline

– **diskonline** *bool* *(**statusdict** dictionary)*

returns a boolean value indicating whether there exists a writeable disk device. The result will be *true* if there exists a device parameter set named %disk*n*% (for some integer value of *n*) in which the value of the **Writeable** parameter is *true*; if no such parameter set exists, a *false* value will be returned. Note that the writeable disk device found need not have an initialized file system.

Errors: **stackoverflow**

diskstatus

– **diskstatus** *free total* *(**statusdict** dictionary)*

returns the current number of free blocks and the total number of blocks available on all writeable disk devices. These values are determined by finding all known device parameter sets named %disk*n*% (for integer values of *n*) in which the value of the **Writeable** parameter is *true*, and totaling the values of their **Free** and **LogicalSize** parameters, respectively (see Table 3.32 on page 61).

Errors: **stackoverflow**

doprinterrors – **doprinterrors** *bool* (***statusdict*** *dictionary)*

returns the current value of the system parameter **DoPrintErrors** (see Table 3.2 on page 15). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setdoprinterrors**

dostartpage – **dostartpage** *bool* (***statusdict*** *dictionary)*

returns the current value of the system parameter **DoStartPage** (see Table 3.2 on page 15). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setdostartpage**

dosysstart – **dosysstart** *bool* (***statusdict*** *dictionary)*

returns a boolean value indicating whether a startup action is to be performed during subsequent restarts of the interpreter. The result will be *true* if the value of the system parameter **StartupMode** (*PLR3*, p. 753) is nonzero and *false* if it is zero. If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setdosysstart**

duplexmode – **duplexmode** *bool* (***statusdict*** *dictionary)*

returns the current value of the page device parameter **Duplex** (*PLR3*, p. 416). If the **Duplex** page device parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setduplexmode**

emulate *stream emulator* **emulate** – (***statusdict*** *dictionary)*
stream dict emulator **emulate** –

causes the PostScript interpreter to yield control to the emulator named by *emulator*. The allowed values of *emulator* are defined by the implicit resource category **Emulator** (see Section 4.2, "Implicit Resource Categories."). An illegal emulation name will cause a **rangecheck** error.

This operator is present only on output devices in which one or more emulators are coresident with the PostScript interpreter. The exact details of such emulators are product-dependent and may vary on different devices, even though the emulation name may be the same. In most coresident emulators, the command sequence Esc Del 0 can be used to return control to the PostScript interpreter; however, the PostScript language context will generally have been lost.

The *stream* operand is a file object that provides the input source for the emulator. The contents of the input stream must be appropriate to the emulator, or an **invalidaccess** error will occur.

The optional *dict* operand is a dictionary holding any parameters the specified emulator may need. If the emulator requires parameters and *dict* is omitted, product-dependent defaults will be used if possible. If the emulator does not require parameters, *dict* will be ignored if present. Currently, no emulators require parameters.

There is no LanguageLevel 2 equivalent for this operator.

Errors: **invalidaccess, rangecheck, stackoverflow, stackunderflow**

firstside – **firstside** *bool* (***statusdict*** *dictionary)*

returns a boolean value indicating whether the current page is to be printed on the front side (*true*) or the back side (*false*) of the output medium.

Note: *This operator is sometimes found on devices that do not support duplex printing. On such devices, it may be used in generating output that is intended to be copied using a duplex copier.*

There is no LanguageLevel 2 equivalent for this operator.

Errors: **stackoverflow**

hardwareiomode – **hardwareiomode** *int* (***statusdict*** *dictionary)*

returns an integer code identifying the currently enabled communication channel (that is, the channel for which the value of the **Enabled** parameter in the corresponding device parameter set is *true*). The possible codes are as follows:

0	%Serial%
1	%Parallel%
2	%LocalTalk%
3	%SerialB%

This operator will always return the channel identified by the system parameter **CurInputDevice** if that channel is on and enabled and is one of the ones listed above. Otherwise, it will return the smallest code in the list denoting a channel that is on and enabled. If no such channel exists, it will return 0.

Errors: **stackoverflow**
See also: **sethardwareiomode**

initializedisk *blocks action* **initializedisk** – (***statusdict*** *dictionary)*

initializes each writeable disk by setting the device parameters **LogicalSize** and **InitializeAction** in each %disk*n*% parameter set (see Table 3.32 on page 61) to the values *blocks* and *action* + 1, respectively.

This operator should be invoked only from within a system administrator job.

Errors: **invalidaccess, ioerror, rangecheck, stackunderflow, typecheck**

jobname – **jobname** *string* (***statusdict*** *dictionary)*

is a string with the same value as the user parameter **JobName** (*PLR3*, p. 750).

jobname is not an operator; it is a name in **statusdict** associated with the job name string. LanguageLevel 1 applications that formerly specified the job name by setting the value of this string, using code such as

```
/jobname (Job 1) def
```

can continue to do so; the PostScript interpreter will automatically update the value of the **JobName** user parameter to match. Similarly, changes in the value of the **JobName** user parameter will affect the value of the **jobname** string.

Errors: **stackoverflow, undefined**

jobtimeout – **jobtimeout** *int* *(**statusdict** dictionary)*

returns the current value of the user parameter **JobTimeout** (see Table 3.1 on page 14). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setjobtimeout**

ledger – **ledger** – *(**userdict** dictionary)*

requests ledger-size paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value of 17 by 11 inches (`[1224 792]` in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing ledger size as the page size whether or not a ledger-size media source is available on the device (see *PLR3*, pp. 434–435).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **ledgertray**

ledgertray – **ledgertray** – *(**statusdict** dictionary)*

requests a paper tray containing ledger-size paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value of 17 by 11 inches (`[1224 792]` in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if a ledger-size media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **ledgertray** operator converts any such **configurationerror** to a **rangecheck** error.

*Note: Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck**
See also: **ledger**

legal – **legal** – *(**userdict** dictionary)*

requests legal-size paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value of 8.5 by 14 inches ([612 1008] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing legal size as the page size whether or not a legal-size media source is available on the device (see *PLR3*, pp. 434–435).

*Note: Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **legaltray**

legaltray – **legaltray** – *(**statusdict** dictionary)*

requests a paper tray containing legal-size paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value of 8.5 by 14 inches ([612 1008] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if a legal-size media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **legaltray** operator converts any such **configurationerror** to a **rangecheck** error.

Note: *Because this operator invokes* **setpagedevice***, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck**
See also: **legal**

letter

– letter – *(**userdict** dictionary)*

requests letter-size paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value of 8.5 by 11 inches (`[612 792]` in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing letter size as the page size whether or not a letter-size media source is available on the device (see *PLR3*, pp. 434–435).

Note: *Because this operator invokes* **setpagedevice***, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **lettersmall, lettertray**

lettersmall

– lettersmall – *(**userdict** dictionary)*

requests letter-size paper as the output medium, with a margin of approximately 1⁄3 inch at all four edges of the physical medium. It invokes the **setpagedevice** operator with a **PageSize** value of 8.5 by 11 inches (`[612 792]` in default user space units) and an **ImagingBBox** value of `[25 25 587 767]` (inset 25 units from each edge of the physical sheet).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing letter size as the page size whether or not a letter-size media source is available on the device (see *PLR3*, pp. 434–435).

Note: *Because this operator invokes* **setpagedevice***, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **letter, lettertray**

lettertray

– **lettertray** – (***statusdict*** *dictionary)*

requests a paper tray containing letter-size paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value of 8.5 by 11 inches (`[612 792]` in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if a letter-size media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **lettertray** operator converts any such **configurationerror** to a **rangecheck** error.

Note: *Because this operator invokes* **setpagedevice***, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck**
See also: **letter, lettersmall**

manualfeed

– **manualfeed** *bool* (***statusdict*** *dictionary)*

is a boolean value that works in conjunction with the page device parameter **ManualFeed** (*PLR3*, pp. 402–403) to determine whether a sheet of input medium is to be fed manually. If the value of either **manualfeed** or **ManualFeed** is *true* at the time of a **showpage** or **copypage** operation, then that page will be fed manually; otherwise, the page will be fed automatically.

manualfeed is not an operator; it is a name in **statusdict** associated with the boolean value. The values of **manualfeed** and **ManualFeed** are independent; setting either value does not affect the other. The initial value of **manualfeed** at power-on is *false*.

Errors: **stackoverflow, undefined**

manualfeedtimeout – **manualfeedtimeout** *int* *(**statusdict** dictionary)*

is an integer that works in conjunction with the page device parameter **ManualFeedTimeout** (see Table 2.2 on page 5) to determine the manual feed timeout interval.

manualfeedtimeout is not an operator; it is a name in **statusdict** associated with the integer timeout value. The values of **manualfeedtimeout** and **ManualFeedTimeout** are independent; setting either value does not affect the other.

By default, **manualfeedtimeout** is not defined and the value of the page device parameter **ManualFeedTimeout** is used to determine the timeout interval. If a job has defined a **manualfeedtimeout** entry in **statusdict**, the value of this entry will be used for the timeout interval in place of the **ManualFeedTimeout** parameter.

Errors: **stackoverflow**

margins – **margins** *top left* *(**statusdict** dictionary)*

returns the values of the top and left page margins, taken from the page device parameter **Margins** (*PLR3*, p. 415). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setmargins**

mirrorprint – **mirrorprint** *bool* *(**statusdict** dictionary)*

returns the current value of the page device parameter **MirrorPrint** (*PLR3*, p. 415). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setmirrorprint**

newsheet – **newsheet** – *(**statusdict** dictionary)*

causes the current page to be printed if it is a back side and the page device parameter **Duplex** (*PLR3*, p. 416) is *true*; otherwise has no effect. The page is printed as is (perhaps blank). The operator then sets up a clean printing environment for the next page. If the **Duplex** page device parameter is not present, an **undefined** error will occur.

There is no LanguageLevel 2 equivalent for this operator (although its effect can be approximated by invoking **setpagedevice** with an empty request dictionary).

Errors: **undefined**
See also: **firstside**

note

– **note** – *(**userdict** dictionary)*

requests the same paper size already established as the current output medium, but with a margin of approximately 1⁄3 inch at all four edges of the physical medium. This operator invokes **setpagedevice** with an **ImagingBBox** value of [25 25 (*width* – 25) (*height* – 25)] (inset 25 units from each edge of the physical sheet), where the dimensions *width* and *height* are taken from the current value of the page device parameter **PageSize**. If the current page size is letter or A4, the **note** operator is equivalent to **lettersmall** or **a4small**, respectively.

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing the specified page size, regardless of whether a media source of that size is available on the device (see *PLR3*, pp. 434–435).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**

pagecount

– **pagecount** *int* *(**statusdict** dictionary)*

returns the current value of the system parameter **PageCount** (*PLR3*, p. 752). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**

pagemargin – **pagemargin** *left* *(**statusdict** dictionary)*

returns the horizontal (x) component of the page device parameter **PageOffset** (*PLR3*, p. 415), measured in default user space units. If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setpagemargin**

pageparams – **pageparams** *width height offset orientation* *(**statusdict** dictionary)*

returns information about the dimensions of the physical medium, the page offset, and the page image orientation that were assumed during the generation of the current page description. The return values *width*, *height*, and *offset* are taken from the page device parameters **PageSize** and **PageOffset** (*PLR3*, pp. 401 and 415, respectively). However, the value returned for *orientation* is determined by device-dependent means and is *not* based on the value of the **Orientation** page device parameter.

If the *orientation* value returned is 0 (normal default orientation), *width* and *height* return the vertical (y) and horizontal (x) components, respectively, of **PageSize**. If *orientation* is 1 (rotated 90 degrees counterclockwise), the dimensions are exchanged, with *width* equal to the x component of **PageSize** and *height* equal to the y component. (Note that the roles of the *width* and *height* operands are the opposite of what the page orientation would seem to imply.) **Orientation** values of 2 (rotated 180 degrees) or 3 (rotated 270 degrees counterclockwise, equivalent to 90 degrees clockwise), as described in *PLR3*, p. 412, are never returned as the value of *orientation*; pages in these orientations are reported as if the **Orientation** value were 0 or 1 instead.

The value returned for *offset* is either the horizontal (x) or vertical (y) component of the **PageOffset** page device parameter, depending on the orientation. The correspondence between the value of *orientation* and the component of **PageOffset** returned is device-dependent and may vary from one output device to another.

Errors: **stackoverflow, undefined**
See also: **setpageparams, setpage**

pagestackorder – **pagestackorder** *bool* *(**statusdict** dictionary)*

returns the logical complement of the page device parameter **OutputFaceUp** (*PLR3*, p. 418)—that is, the value returned for *bool* is *true* if **OutputFaceUp** is *false* and *false* if **OutputFaceUp** is *true*. If the **OutputFaceUp** parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setpagestackorder**

printername *string* **printername** *substring* *(**statusdict** dictionary)*

copies the value of the system parameter **PrinterName** (*PLR3*, p. 753) into *string* and returns a string object designating the substring of *string* actually used. If the **PrinterName** parameter is not present, an **undefined** error will occur.

Errors: **rangecheck, stackunderflow, typecheck, undefined**
See also: **setprintername**

processcolors – **processcolors** *int* *(**statusdict** dictionary)*

returns the number of color components in the current page device's process color model (1 for **DeviceGray**, 3 for **DeviceRGB** or **DeviceCMY**, 4 for **DeviceCMYK** or **DeviceRGBK**, or the number of colorants specified in a **DeviceN** color model; see *PLR3*, pp. 422–424).

***Note:** This operator is mandatory on devices that can produce more than one color; it is optional (and typically is not present) on monochrome devices. Its absence indicates a monochrome device (one process color).*

Errors: **stackoverflow**

product – **product** *string* *(**statusdict** dictionary)*

is a string object representing the name of the product on which the PostScript interpreter is running. **product** is not an operator; it is a name in **statusdict** associated with the string object. This string is initialized to the same value returned by the standard **systemdict** operator **product** (*PLR3*, p. 634).

Errors: **stackoverflow**

ramsize – **ramsize** *int* *(**statusdict** dictionary)*

returns the number of bytes of installed RAM available to the page description language, taken from the **RamSize** system parameter (see Table 3.2 on page 15). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**

realformat – **realformat** *string* *(**statusdict** dictionary)*

is a read-only string with the same value as the system parameter **RealFormat** (*PLR3*, p. 753). **realformat** is not an operator; it is a name in **statusdict** associated with the string object.

Errors: **stackoverflow**

resolution – **resolution** *pixelsperinch* *(**statusdict** dictionary)*

returns the horizontal resolution of the current output device in pixels per inch, taken from the horizontal (*x*) component of the **HWResolution** page device parameter (*PLR3*, p. 414). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **setresolution**

revision – **revision** *int* *(**statusdict** dictionary)*

is a read-only integer with the same value as the system parameter **Revision** (*PLR3*, p. 753). **revision** is not an operator; it is a name in **statusdict** associated with the integer value.

Errors: **stackoverflow**

sccbatch *channel* **sccbatch** *baud options* (***statusdict*** *dictionary)*

returns the values of device parameters for serial communication. The *channel* operand identifies the parameter set affected:

channel	PARAMETER SET
9	%SerialB_NV%
25	%Serial_NV%

baud returns the value of the **Baud** device parameter; *options* is a 1-byte integer (0–255) that encodes the values of the **StopBits**, **DataBits**, **FlowControl**, and **Parity** parameters as equivalent LanguageLevel 1 settings. (See Table 3.4 on page 33 for descriptions of all of these device parameters.) The settings are encoded as follows, where the bit positions within the *options* byte are numbered from 7 (high-order) to 0 (low-order):

BIT 7	STOP BITS	BITS 6–5	DATA BITS	BITS 4–3	FLOW CONTROL	BITS 1–0	PARITY
0	1 stop bit	0	Standard	0	XON/XOFF	0	Space
1	2 stop bits	1	7 bits	1	DTR	1	Odd
		2	8 bits	2	ETX/ACK	2	Even
						3	Mark

The value of the LanguageLevel 2 **StopBits** parameter maps directly into the equivalent LanguageLevel 1 settings shown above. For the **FlowControl** parameter, the values XonXoff, XonXoff2, and RobustXonXoff all map to an *options* setting of 0 (XON/XOFF), Dtr or DtrLow to 1 (DTR), and EtxAck to 2 (ETX/ACK).

For the **DataBits** and **Parity** parameters, the correspondence is less straightforward:

LANGUAGELEVEL 2		LANGUAGELEVEL 1	
DataBits	Parity	DATA BITS	PARITY
7	None	7 bits	Mark
	Space	7 bits	Space
	Mark	7 bits	Mark
	Odd	7 bits	Odd
	Even	7 bits	Even
8	None	8 bits	Mark
	Space	8 bits	Space
	Mark	8 bits	Mark
	Odd	8 bits	Odd
	Even	8 bits	Even

Note: *These mappings are intended to provide the best attainable compatibility with LanguageLevel 1 behavior, but in some cases no exact equivalent is possible. For ex-*

ample, LanguageLevel 1 does not support the combination of 7 data bits with no parity (that is, 7 bits total for data and parity together). Thus the combination of ***DataBits*** *equal to 7 with* ***Parity*** *equal to None is imperfectly mapped to 7 data bits with mark parity.*

Errors: rangecheck, stackoverflow, stackunderflow, typecheck
See also: setsccbatch

sccinteractive *channel* **sccinteractive** *baud options* (***statusdict*** *dictionary)*

formerly returned the values of device parameters for serial communication in interactive or emulation mode. This operator now pops the *channel* operand off the stack, pushes zero values on the stack for *baud* and *options*, and otherwise does nothing.

There is no LanguageLevel 2 equivalent for this operator.

Errors: invalidaccess, rangecheck, stackoverflow, stackunderflow, typecheck
See also: setsccinteractive

setaccuratescreens *bool* **setaccuratescreens** – (***statusdict*** *dictionary)*

sets the value of the user parameter **AccurateScreens** (*PLR3*, pp. 749 and 757) to *bool*. If this parameter is not present, **setaccuratescreens** does nothing.

Errors: invalidaccess, stackunderflow, typecheck, undefined
See also: accuratescreens

setdefaulttimeouts *job manualfeed wait* **setdefaulttimeouts** – (***statusdict*** *dictionary)*

sets the values of the system parameters **JobTimeout** and **WaitTimeout** (see Table 3.2 on page 15), and the page device parameter **ManualFeedTimeout** (Table 2.2 on page 5) to *job*, *wait*, and *manualfeed*, respectively.

This operator always accepts three operands, even if one or more of the relevant system or page device parameters are not present. If **JobTimeout** or **WaitTimeout** is absent, the corresponding operand is ignored; if **ManualFeedTimeout** is absent, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

This operator should be invoked only from within a system administrator job.

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3*, pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: invalidaccess, rangecheck, stackunderflow, typecheck
See also: defaulttimeouts

setdoprinterrors

bool **setdoprinterrors** – (***statusdict*** *dictionary)*

sets the value of the system parameter **DoPrintErrors** (see Table 3.2 on page 15) to *bool*. If this parameter is not present, **setdoprinterrors** does nothing.

This operator should be invoked only from within a system administrator job.

Errors: invalidaccess, stackunderflow, typecheck, undefined
See also: doprinterrors

setdostartpage

bool **setdostartpage** – (***statusdict*** *dictionary)*

sets the value of the system parameter **DoStartPage** (see Table 3.2 on page 15) to *bool*. If this parameter is not present, **setdostartpage** does nothing.

This operator should be invoked only from within a system administrator job.

Errors: invalidaccess, stackunderflow, typecheck, undefined
See also: dostartpage

setdosysstart

bool **setdosysstart** – (***statusdict*** *dictionary)*

sets the value of the system parameter **StartupMode** (*PLR3*, p. 753) according to the value of *bool:* to 1 if *bool* is *true* and 0 if it is *false*. If this parameter is not present, **setdosysstart** does nothing.

This operator should be invoked only from within a system administrator job.

Errors: invalidaccess, stackunderflow, typecheck, undefined
See also: dosysstart

setduplexmode *bool* **setduplexmode** – (***statusdict*** *dictionary)*

sets the value of the page device parameter **Duplex** (*PLR3*, p. 416) to *bool*. If this parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3*, pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: stackunderflow, typecheck, undefined
See also: duplexmode

sethardwareiomode *int* **sethardwareiomode** – (***statusdict*** *dictionary)*

opens the specified channel(s) for communication. The operand *int* is a code identifying the channels to be opened:

0	%Serial% and %SerialB%
1	%Parallel%
2	%LocalTalk% or %EtherTalk% (or both if they exist)
3	%Serial% and %SerialB%

The specified channels are opened by setting their **On** and **Enabled** parameters (see Table 3.3 on page 29) to *true*; all other channels are explicitly closed by setting their **On** and **Enabled** parameters to *false*.

This operator should be invoked only from within a system administrator job.

Errors: invalidaccess, rangecheck, stackunderflow, typecheck
See also: hardwareiomode

setjobtimeout *int* **setjobtimeout** – (***statusdict*** *dictionary)*

sets the value of the user parameter **JobTimeout** (see Table 3.1 on page 14) to *int*. If this parameter is not present, **setjobtimeout** does nothing.

Errors: stackunderflow, typecheck, undefined
See also: jobtimeout

setmargins *top left* **setmargins** – *(**statusdict** dictionary)*

sets the value of the page device parameter **Margins** (*PLR3*, p. 415) to [*left top*]. If this parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

This operator should be invoked only from within a system administrator job.

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **invalidaccess, rangecheck, stackunderflow, typecheck, undefined**
See also: **margins**

setmirrorprint *bool* **setmirrorprint** – *(**statusdict** dictionary)*

sets the value of the page device parameter **MirrorPrint** (*PLR3*, p. 415) to *bool*. If this parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **stackunderflow, typecheck**
See also: **mirrorprint**

setpage *width height orientation* **setpage** – *(**statusdict** dictionary)*

sets the values of the page device parameters **PageSize** and **Orientation** (*PLR3*, pp. 401 and 412, respectively) as specified by the operands. If either of these parameters is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

The value supplied for *orientation* must be either 0 (normal default orientation) or 1 (rotated 90 degrees clockwise). The page device parameter **Orientation** is set to a corresponding value appropriate to the particular output device. (The exact correspondence between the value of the *orientation* operand that of the **Orientation** page device parameter is device-dependent and may vary from one output device to another.) Values of *orientation* corresponding to **Orientation** settings of 2 (rotated 180 degrees) or 3 (rotated 270 degrees counterclockwise, equivalent to 90 degrees clockwise) are not supported.

The **PageSize** page device parameter is set to [*height width*] if the value of *orientation* is 0, or to [*width height*] if *orientation* is 1. (Note that the roles of the *width* and *height* operands are the opposite of what the page orientation would seem to imply.)

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck, stackunderflow, typecheck**
See also: **pageparams, setpageparams**

setpagemargin

left **setpagemargin** – (***statusdict** dictionary*)

sets the value of the page device parameter **PageOffset** (*PLR3*, p. 415) to [*left* 0]. If this parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **rangecheck, stackunderflow, typecheck**
See also: **pagemargin**

setpageparams

width height offset orientation **setpageparams** – (***statusdict** dictionary*)

sets the values of the page device parameters **PageSize**, **PageOffset**, and **Orientation** (*PLR3*, pp. 401, 415, and 412, respectively) as specified by the operands. If any of these parameters is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

The value supplied for *orientation* must be either 0 (normal default orientation) or 1 (rotated 90 degrees clockwise). The page device parameter **Orientation** is set to a corresponding value appropriate to the particular output device. (The exact correspondence between the value of the *orientation* operand that of the **Orientation** page device parameter is device-dependent and may vary from one output device to another.) Values of *orientation* corresponding to **Orientation** settings of 2 (rotated 180 degrees) or 3 (rotated 270 degrees counterclockwise, equivalent to 90 degrees clockwise) are not supported.

The **PageSize** page device parameter is set to [*height width*] if the value of *orientation* is 0, or to [*width height*] if *orientation* is 1. (Note that the roles of the *width* and *height* operands are the opposite of what the page orientation would seem to imply.) Either the horizontal (*x*) or the vertical (*y*) component of the **PageOffset** page device parameter is set to the value of the *offset* operand; the choice of which component to set is device-dependent.

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck, stackunderflow, typecheck, undefinedresult**
See also: **pageparams, setpage**

setpagestackorder

bool **setpagestackorder** – (***statusdict*** *dictionary)*

sets the value of the page device parameter **OutputFaceUp** (*PLR3*, p. 418) to the logical complement of *bool*—that is, to *true* if *bool* is *false* and to *false* if *bool* is *true*. If the **OutputFaceUp** page device parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

This operator should be invoked only from within a system administrator job.

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **invalidaccess, stackunderflow, typecheck, undefined**
See also: **pagestackorder**

setprintername

string **setprintername** – (***statusdict*** *dictionary)*

sets the value of the system parameter **PrinterName** (*PLR3*, p. 753) to *string*. If this parameter is not present, **setprintername** does nothing.

This operator should be invoked only from within a system administrator job.

Errors: **invalidaccess, limitcheck, stackunderflow, typecheck, undefined**
See also: **printername**

setresolution

pixelsperinch **setresolution** – (***statusdict*** *dictionary)*

sets the value of the page device parameter **HWResolution** (*PLR3*, p. 414) to [*pixelsperinch pixelsperinch*]. If this parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: rangecheck, stackunderflow, typecheck
See also: resolution

setsccbatch

channel baud options **setsccbatch** – (***statusdict*** *dictionary)*

sets the values of device parameters for serial communication. The *channel* operand identifies the parameter set affected:

channel	PARAMETER SET
9	%SerialB_NV%
25	%Serial_NV%

The device parameters affected are **Baud**, **StopBits**, **DataBits**, **FlowControl**, **Parity**, and **CheckParity** (see Table 3.4 on page 33).

baud specifies the value of the **Baud** device parameter. *options* is a 1-byte integer (0–255) that encodes the LanguageLevel 1 settings for stop bits, data bits, flow control, and parity. The settings are specified as follows, where the bit positions within the *options* byte are numbered from 7 (high-order) to 0 (low-order):

BIT 7	STOP BITS	BITS 6–5	DATA BITS	BITS 4–3	FLOW CONTROL	BITS 1–0	PARITY
0	1 stop bit	0	Standard	0	XON/XOFF	0	Space
1	2 stop bits	1	7 bits	1	DTR	1	Odd
		2	8 bits	2	ETX/ACK	2	Even
						3	Mark

The LanguageLevel 1 settings shown for stop bits and flow control map directly to corresponding LanguageLevel 2 values of the **StopBits** and **FlowControl** device parameters, but the mapping for data bits and parity is less straightforward:

LANGUAGELEVEL 1		LANGUAGELEVEL 2	
DATA BITS	**PARITY**	**DataBits**	**Parity**
Standard	Space	7 bits	Space
	Mark	8 bits	None
	Odd	7 bits	Odd
	Even	7 bits	Even
7 bits	Space	7 bits	Space
	Mark	7 bits	Mark
	Odd	7 bits	Odd
	Even	7 bits	Even
8 bits	Space	8 bits	None
	Mark	8 bits	None
	Odd	8 bits	Odd
	Even	8 bits	Even

***Note:** Most serial hardware does not support the combinations 8-bit mark or 8-bit space, so these combinations of parameter values are never generated in LanguageLevel 2. (In fact, in LanguageLevel 1, these combinations behave equivalently to 8-bit* None *in LanguageLevel 2.)*

The **CheckParity** parameter is set according to the new **Parity** setting:

- *true* if **Parity** is Odd or Even.
- *false* if **Parity** is Space or Mark.
- Not changed if **Parity** is None. (Parity checking is not done if **Parity** is independent of the setting of **CheckParity**.)

This operator should be invoked only from within a system administrator job.

Errors: invalidaccess, rangecheck, stackunderflow, typecheck
See also: sccbatch

setsccinteractive

channel baud options **setsccinteractive** – *(**statusdict** dictionary)*

formerly set the values of device parameters for serial communication in interactive or emulation mode. This operator now pops its three operands off the stack and otherwise does nothing.

This operator should be invoked only from within a system administrator job.

There is no LanguageLevel 2 equivalent for this operator.

Errors: **invalidaccess, rangecheck, stackunderflow, typecheck**
See also: **sccinteractive**

setsoftwareiomode

int **setsoftwareiomode** – (***statusdict*** *dictionary)*

sets the values of the device parameters **Interpreter** and, if appropriate, **Protocol** (see Tables 3.3 and 3.4 on pages 29 and 33) in the parameter set designated by the system parameter **CurInputDevice** (Table 3.2 on page 15). The *int* operand is a code specifying the values for **Interpreter** and **Protocol**, as follows:

int	Interpreter	Protocol
0	PostScript	Normal
1	ProprinterXL	Raw
2	Diablo630	Raw
3	(Reserved)	
4	HP7475A	Raw
5	LaserJetIIP or LaserJetIII (whichever is installed)	Raw
100	PostScript	Binary

Note: *In the unlikely event that both the LaserJetIIP and LaserJetIII emulators are installed, an operand value of 5 will select only LaserJetIIP.*

This operator should be invoked only from within a system administrator job.

Errors: **invalidaccess, rangecheck, stackunderflow, typecheck**
See also: **softwareiomode**

settumble

bool **settumble** – (***statusdict*** *dictionary)*

sets the value of the page device parameter **Tumble** (*PLR3*, p. 416) to *bool*. If this parameter is not present, the **Policies** dictionary is consulted and the applicable recovery policy is invoked (see *PLR3*, Section 6.2.7).

Note: *Because this operator invokes* **setpagedevice**, *it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **stackunderflow, typecheck**
See also: **tumble**

setuserdiskpercent *int* **setuserdiskpercent** – (***statusdict*** *dictionary)*

formerly set the percentage of disk space reserved for user files. This operator now pops the operand *int* off the stack and otherwise does nothing.

This operator should be invoked only from within a system administrator job.

There is no LanguageLevel 2 equivalent for this operator.

Errors: **invalidaccess, rangecheck, stackunderflow, typecheck**
See also: **userdiskpercent**

softwareiomode – **softwareiomode** *int* (***statusdict*** *dictionary)*

returns a code representing the current value of the device parameter **Interpreter** (see Table 3.3 on page 29) in the parameter set designated by the system parameter **CurInputDevice** (Table 3.2 on page 15). See the description of the **setsoftwareiomode** compatibility operator for possible result values and their meanings.

Note: *If the value of* ***Interpreter*** *is not one of those listed under* ***setsoftwareiomode****, the int value returned will be -1.*

Errors: **stackoverflow**
See also: **setsoftwareiomode**

tumble – **tumble** *bool* (***statusdict*** *dictionary)*

returns the current value of the page device parameter **Tumble** (*PLR3*, p. 416). If this parameter is not present, an **undefined** error will occur.

Errors: **stackoverflow, undefined**
See also: **settumble**

userdiskpercent – **userdiskpercent** *int* (***statusdict*** *dictionary)*

formerly returned the percentage of disk space reserved for user files. This operator now does nothing and returns the value 0.

There is no LanguageLevel 2 equivalent for this operator.

Errors: **stackoverflow**
See also: **setuserdiskpercent**

waittimeout – **waittimeout** *int* *(**statusdict** dictionary)*

is an integer with the same value as the user parameter **WaitTimeout** (see Table 3.1 on page 14).

waittimeout is not an operator; it is a name in **statusdict** associated with the integer value of the wait timeout. LanguageLevel 1 applications that formerly specified the wait timeout by setting this value, using code such as

```
/waittimeout 120 def
```

can continue to do so; the PostScript interpreter will automatically update the value of the **WaitTimeout** user parameter to match. Similarly, changes in the value of the **WaitTimeout** user parameter will affect the value of **waittimeout**.

Errors: **stackoverflow, undefined**

11x17 – **11x17** – *(**userdict** dictionary)*

requests 11-by-17-inch paper as the output medium. It invokes the **setpagedevice** operator with a **PageSize** value of 11 by 17 inches ([792 1224] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 7, disabling the normal LanguageLevel 2 media selection mechanism and unconditionally establishing 11 by 17 inches as the page size whether or not an 11-by-17-inch media source is available on the device (see *PLR3*, pp. 434–435).

***Note:** Because this operator invokes **setpagedevice**, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck**
See also: **11x17tray**

11x17tray – **11x17tray** – (***statusdict*** *dictionary)*

requests a paper tray containing 11-by-17-inch paper as the media source. It invokes the **setpagedevice** operator with a **PageSize** value of 11 by 17 inches ([792 1224] in default user space units) and an **ImagingBBox** value of *null*, denoting the largest bounding box possible for this page size on the current output device (see *PLR3*, p. 414).

This operator sets the **PageSize** entry in the **Policies** dictionary to 0, causing a **configurationerror** to be raised if an 11-by-17-inch media source is not available on the device (see *PLR3*, p. 434). For compatibility with LanguageLevel 1, the implementation of the **11x17tray** operator converts any such **configurationerror** to a **rangecheck** error.

Note: *Because this operator invokes* ***setpagedevice****, it also causes all of that operator's attendant side effects, such as reinitializing the graphics state and erasing the current page; see* PLR3, *pp. 679–680 and Section 6.2.6 (pp. 426–432).*

Errors: **limitcheck, rangecheck**
See also: **11x17**

APPENDIX A

Changes since Earlier Versions

A.1 Changes since Version 3010

The following changes have been made to the PostScript language since version 3010. These changes were incorporated into the 3011 release.

- A new kind of Type 0 CIDFont has been added: a CFF CIDFont, defined in **FontSet** resources rather than in a CIDFont file. See Section 5.1.2, “CFF CIDFonts.”
- The Type 2 CIDFont dictionary has been extended to accommodate incremental downloading of CJK TrueType CIDFonts. See Section 5.1.3, “Incremental Downloading of CJK TrueType CIDFonts.”
- A new optional entry, **InterpolationDetails**, has been added to all image dictionaries, regardless of type. See Section 5.2, “Interpolation Details Dictionary.”
- A new optional trapping parameter, **ImagemaskTrapping**, has been added to the trapping parameter dictionary. See Section 5.3, “Imagemask Trapping Parameter.”

APPENDIX B

Implementation Limits

THIS APPENDIX SUPPLEMENTS Appendix B of the *PostScript Language Reference*, Third Edition. It provides additional information about limits that are typical of PostScript interpreter implementations from Adobe Systems.

B.1 Columns Limit in CCITTFaxDecode

The **Columns** entry in the **CCITTFaxDecode** dictionary (*PLR3*, pp. 144-145) is limited to a maximum value of 62,000.

APPENDIX C

Emulator Parameter Sets

PARAMETER SETS OF TYPE Emulator provide supporting information for emulating non-PostScript imaging systems within a PostScript imaging system.

An *emulator* is an interpreter for input stream data that is formatted in a page description language (PDL) other than PostScript. Such a PDL would originally be used on a non-PostScript imaging system. By incorporating PDL emulators, PostScript imaging systems can emulate other imaging systems.

An imaging system needs to know how to interpret the stream of input characters on its data channels in order make marks on the page. The rules it uses are specified by the **Interpreter** device parameter, which is part of each Communications parameter set that is defined for the data channels. (For more information, see Section 3.3.3, "Communications Parameter Sets," and in particular the description of the **Interpreter** parameter in Table 3.3 on page 29.) If the value of **Interpreter** is other than PostScript, the imaging system is being asked to emulate the functions of some other imaging system.

Emulator parameter sets were originally introduced to communicate to an emulation interpreter the values of job-level parameters that might change from one print job to another. They also provided a means of querying and changing the current value of a parameter. For example, a **Copies** parameter would indicate how many copies of a page to print; the number desired could be changed by setting the parameter.

Job-level control parameters required by non-PostScript imaging systems often varied among products using the same interpreter PDL. As a result, the term *emulator* came to refer, in the context of emulator parameter sets, to the imaging system itself, rather than to the interpreter PDL.

This situation changed when PCL (Printer Control Language) interpreter imaging systems introduced the Printer Job Language (PJL) to support job-level control parameters. PJL replaced the need to rely on emulator parameter sets for this purpose. However, emulator parameter sets are still useful in some cases. A particular imaging system product might use them for its own emulator purposes. They might also be used to share a parameter between PostScript and an emulator.

As a side effect of the introduction of PJL, the use of the term *emulator* also changed (in the context of Emulator parameter sets) from referring to a particular imaging system to referring to the interpreter PDL being emulated. Note that with the introduction of the **PDL** resource category, any desired information about a specific imaging system target or version of a PDL being emulated is appropriately provided using this resource.

PCL is an emulator that is included in many PostScript imaging systems. The %PCL% parameter set applies to systems that include an emulator for one or more versions of PCL. The required entries in the %PCL% parameter set are described in Table C.1.

TABLE C.1 Device parameters in the %PCL% parameter set

KEY	TYPE	VALUE
Type	name	*(Read-only)* The parameter set type; must be Emulator.
MaxPermanentStorage	integer	The maximum amount of memory (in bytes) to be dedicated for use by the PCL emulator.

APPENDIX D

Typical Font Set

THIS APPENDIX SUPPLEMENTS Appendix E of the *PostScript Language Reference*, Third Edition. It lists the typical set of fonts available on most Adobe PostScript 3 products. While there is no standard set of required fonts, Adobe PostScript 3 products typically have the 136 fonts listed below built into the interpreter.

AlbertusMT
AlbertusMT-Italic
AlbertusMT-Light
AntiqueOlive-Roman
AntiqueOlive-Italic
AntiqueOlive-Bold
AntiqueOlive-Compact
Apple-Chancery
ArialMT
Arial-ItalicMT
Arial-BoldMT
Arial-BoldItalicMT
AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bodoni
Bodoni-Italic
Bodoni-Bold
Bodoni-BoldItalic
Bodoni-Poster
Bodoni-PosterCompressed
Bookman-Light
Bookman-LightItalic
Bookman-Demi
Bookman-DemiItalic
Carta
Chicago
Clarendon
Clarendon-Light
Clarendon-Bold
CooperBlack
CooperBlack-Italic
Copperplate-ThirtyTwoBC
Copperplate-ThirtyThreeBC
Coronet-Regular
Courier
Courier-Oblique
Courier-Bold
Courier-BoldOblique
Eurostile
Eurostile-Bold
Eurostile-ExtendedTwo
Eurostile-BoldExtendedTwo
Geneva
GillSans

GillSans-Italic
GillSans-Bold
GillSans-BoldItalic
GillSans-Condensed
GillSans-BoldCondensed
GillSans-Light
GillSans-LightItalic
GillSans-ExtraBold
Goudy
Goudy-Italic
Goudy-Bold
Goudy-BoldItalic
Goudy-ExtraBold
Helvetica
Helvetica-Oblique
Helvetica-Bold
Helvetica-BoldOblique
Helvetica-Condensed
Helvetica-Condensed-Oblique
Helvetica-Condensed-Bold
Helvetica-Condensed-BoldObl
Helvetica-Narrow
Helvetica-Narrow-Oblique
Helvetica-Narrow-Bold
Helvetica-Narrow-BoldOblique
HoeflerText-Regular
HoeflerText-Italic
HoeflerText-Black
HoeflerText-BlackItalic
HoeflerText-Ornaments
JoannaMT
JoannaMT-Italic
JoannaMT-Bold
JoannaMT-BoldItalic
LetterGothic
LetterGothic-Slanted
LetterGothic-Bold
LetterGothic-BoldSlanted
LubalinGraph-Book
LubalinGraph-BookOblique
LubalinGraph-Demi
LubalinGraph-DemiOblique
Marigold
Monaco
MonaLisa-Recut
NewCenturySchlbk-Roman
NewCenturySchlbk-Italic
NewCenturySchlbk-Bold
NewCenturySchlbk-BoldItalic
NewYork
Optima
Optima-Italic
Optima-Bold
Optima-BoldItalic
Oxford
Palatino-Roman
Palatino-Italic
Palatino-Bold
Palatino-BoldItalic
StempelGaramond-Roman
StempelGaramond-Italic
StempelGaramond-Bold
StempelGaramond-BoldItalic
Symbol
Tekton
Times-Roman
Times-Italic
Times-Bold
Times-BoldItalic
TimesNewRomanPSMT
TimesNewRomanPS-ItalicMT
TimesNewRomanPS-BoldMT
TimesNewRomanPS-BoldItalicMT
Univers
Univers-Oblique
Univers-Bold
Univers-BoldOblique
Univers-Light
Univers-LightOblique
Univers-Condensed

Univers-CondensedOblique	Univers-BoldExt
Univers-CondensedBold	Univers-BoldExtObl
Univers-CondensedBoldOblique	Wingdings-Regular
Univers-Extended	ZapfChancery-MediumItalic
Univers-ExtendedObl	ZapfDingbats

APPENDIX E

Updates to the *PostScript Language Reference,* Third Edition

THIS APPENDIX DESCRIBES changes, additions, and errata to the *PostScript Language Reference*, Third Edition. The changes described here apply to all implementations of LanguageLevel 3, and are limited to additions, corrections, and clarifications to material in *PLR3*; they do not include any changes in the PostScript language that have been introduced as extensions to LanguageLevel 3.

E.1 Errata for the First Printing

This section gives errata (not yet published) to the first printing of the *PostScript Language Reference*, Third Edition. This is the only printing that has appeared up to the time of publication of this *Supplement*.

E.1.1 Chapter 4 Errata

The following change should be made in Chapter 4:

- Page 212, Figure 4.5. There is an extraneous horizontal line between **CIEBasedDEFG** and **DeviceRGB**.

E.1.2 Chapter 5 Errata

The following change should be made in Chapter 5:

- Page 344, Example 5.7. Change %%DocumentNeedResources to %%DocumentNeededResources.

E.1.3 Chapter 6 Errata

The following changes should be made in Chapter 6:

- Page 408, last line before the code example. Change "a sheet of special media" to "a sheet of special medium."
- Page 427. Insert following the first paragraph after the table:

***Note:** The page device's default **Install** procedure establishes the default device-dependent graphics state parameters that are appropriate for the device. If a PostScript program replaces the **Install** procedure, the replacement procedure should call the original procedure as part of its execution; otherwise, some parameters may not be initialized properly.*

- Pages 447–449, Table 6.13. The following entry was omitted from the table:

KEY	TYPE	VALUE
HalftoneName	name or null	*(Optional)* The name of a **Halftone** resource with which to mark traps. It is recommended that the resource be defined in global VM. The **Halftone** resource may be used at unpredictable times, including at a **save** level lower than the current one. If this parameter is null or undefined, traps will be marked using the halftone in effect at the time, possibly causing unexpected results.

E.1.4 Chapter 8 Errata

The following changes should be made in Chapter 8:

- Page 552, **cshow**. Add **stringwidth** and **charpath** to the "See Also" list. (This is to indicate that the special glyph selection process applies to these operators as well as to the ones having the word **show** in their names.)
- Page 557, **currentfont**. Append the following paragraph:

***Note:** As discussed in Section 5.10.3, "Nested Composite Fonts," the **makefont**, **scalefont**, and **selectfont** operators do not apply their transformations recursively to the descendant base fonts or CIDFonts of a composite font. When the current font is such an untransformed descendant of a transformed composite font, **currentfont** returns a copy of the descendant font whose **FontMatrix** has been appropriately transformed to reflect the transformation previously performed on the parent composite font.*

- Page 620, **languagelevel**. Change "returns an integer designating…" to "is an integer designating…." At the end of the operator description, add the sentence "**languagelevel** is not an operator; it is a name in **systemdict** associated with the integer."
- Page 716, **version**. Change "returns a string that identifies…" to "is a read-only string object that identifies…." At the end of the operator description, add the sentence "**version** is not an operator; it is a name in **systemdict** associated with the string."

E.1.5 Bibliography Errata

The following changes should be made in the Bibliography:

- Page 811, following first paragraph. The correct URL for the Adobe Developer Relations Web site is now:

  <http://partners.adobe.com/asn/developer/>

- Page 814, fifth paragraph. Change to read as follows:

International Organization for Standardization (ISO), ISO/IEC 10918-1, *Digital Compression and Coding of Continuous-Tone Still Images*. Informally known as the JPEG standard, for the Joint Photographic Experts Group (the organization that developed the standard). May be ordered from ISO at <http://www.iso.ch>.

Bibliography

SOME DOCUMENTS LISTED IN THIS BIBLIOGRAPHY are indicated as being available on the Adobe Developer Relations site on the World Wide Web. This site is located at

<http://partners.adobe.com/asn/developer/>

Document version numbers given in this Bibliography are the latest as of the time of publication; more recent versions may be found on the Web site.

Resources from Adobe Systems Incorporated

PostScript Language Reference, Third Edition, Addison-Wesley, Reading, MA, 1999. Referred to throughout this *Supplement* as *PLR3*.

PostScript Language Program Design, Addison-Wesley, Reading, MA, 1988. Though this edition of this book describes LanguageLevel 1 only, it is still useful for programmers interested in the effective and efficient design of PostScript programs and printer drivers.

Technical Notes. The following Technical Notes are available on the Adobe Developer Relations Web site:

- *Adobe Serial and Parallel Communications Protocols Specification*, Technical Note #5009
- *Adobe SCSI Input Protocol Specification Version 1.0*, Technical Note #5010

Other Resources

Hewlett-Packard, *Printer Job Language Technical Reference*, part of the Technical Reference Bundle, Part Number 5021-0377, which may be ordered from <http://www.hp.com>.

Institute of Electrical and Electronics Engineers, Inc. (IEEE), Standard 1284-1994, *Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers*. May be ordered from <http://www.ieee.com>.

International Color Consortium, *ICC Profile Format Specification*. This specification and related documents are available at <http://www.color.org>.

International Organization for Standardization (ISO), ISO 639, *Codes for the Representation of Names of Languages*; ISO 3166, *Codes for the Representation of Names of Countries and Their Subdivisions*; and ISO/IEC 8824-1, *Abstract Syntax Notation One (ASN.1): Specification of basic notation*. May be ordered from ISO at <http://www.iso.ch>.

Internet Engineering Task Force (IETF) Request for Comments (RFC) 1157, *A Simple Network Management Protocol*. Available through <http://www.rfc-editor.org>.

Malamud, C., *Analyzing Novell Networks*, Van Nostrand Reinhold, New York, 1992. A discussion of Novell networks begins on page 81.

Sidhu, G. S., Andrews, R. F., and Oppenheimer, A. B., *Inside AppleTalk*, Addison-Wesley, Reading, MA, 1990.

Tanenbaum, A. S., *Computer Networks*, Prentice-Hall, Inc., New Jersey, 1996. Describes the TokenRing and Ethernet protocols.

Index