3.3 Release Notes: Window Server

This file contains release notes for the 3.3, 3.2, 3.1, and 3.0 releases of the Window Server. Items specific to the 3.2 release are listed first, and the Release 3.1 and 3.0 notes follow. There are no items specific to the 3.3 release.

Notes Specific to Release 3.2

New Features in 3.2

The performance of various imaging and window-moving operatations have been improved for both Motorola and Intel implementations of NEXTSTEP. Among these improvements are the following:

- In 16-bit color windows, a **compositerect** operation of type **Sover** is 4 times faster when the color isn't dithered (that is, when the premultiplied RGBA

components are each exactly expressible in 4 bits).

- 8-bit grayscale **Sover**'ing now avoids multiplications when the source alpha is 0 or 1. Performance of both 8-bit grayscale and 16-bit **Sover**'ing have been improved.

- Window-dragging of buffered windows is now done by copying from the backing store to the screen (DRAM€->€VRAM) rather than from screen to screen (VRAM€->€VRAM). This improves window dragging performance on systems with slow VRAM access times.

- 8-bit and 16-bit identity case imaging operations are now faster.

- Overall performance of 32-bit color drawing has been improved.

Bugs Fixed in Release 3.2

These bugs have been fixed in Release 3.2:

Reference	37336
Problem	Meshed 8-bit grayscale images with transparency were improperly displayed.
Description	On 8-bit grayscale or 24-bit color NEXTSTEP systems, images

defined as meshed 8-bit grayscale displayed incorrectly if they had any transparency.

Reference 36064

Problem **readimage** didn't work properly on 32-bit color windows on the i386 architecture.

Description The value returned by **readimage** on an i386 NEXTSTEP system would return non-byteswapped data if used on a 32-bit color window.

Notes Specific to Release 3.1

New Features in 3.1

The rendering algorithm for DeviceCMYK colors on RGB devices has been improved.

setcmykadjust *boolean* **setcmykadjust** -

When set to true, **setcmykadjust** invokes an enhanced algorithm for rendering /**DeviceCMYK** colors on an RGB device. This algorithm is computationally more

expensive that the simple algorithm in the Red Book, but results in greater accuracy for publishing applications. Similar to **setstrokeadjust**, the current value of **setcmykadjust** is stored in the graphics state.

When **setcmykadjust** is active, the behavior of **currentcmykcolor**, **currentgray**, **currentrgbcolor**, and **currenthsbcolor** changes so that any conversions in or out of the **DeviceCMYK** colorspace on behalf of these operators produces a result which if passed to the appropriate **...setcolor** operator yields a visually identical color. In other words, **current...color** followed by **set...color** should not change the visual appearance of the current color . In certain cases, however, an identical match is not possible. For example, **currentcmykcolor** may not be able to return a visually identical cmyk color if the current color has been specified in the **DeviceRGB** colorspace. This is because some **DeviceRGB** colors will fall outside of the gamut of the **DeviceCMYK** approximation used by **setcmykadjust**. In these cases, **currentcmykcolor** returns a reasonably similar color.

cmykadjust normally defaults to true for windowdevices. However, this can be overiden on a per-context basis by using the **setdefaultcmykadjust** operator.

currentcmykadjust - **currentcmykadjust** *boolean*

currentcmykadjust returns a boolean indicating whether or not cmyk adjustment is active for the current graphics state.

setcmykadjustparams *[Cr Cg Cb Mr Mg Mb Yr Yg Yb Rr Rg Rb Gr Gg Gb Br Bg Bb]*
gamma **setcmykadjustparams -**

The cmyk to rgb algorithm invoked by **setcmykadjust** is based on a simple six color Neugebauer approximation. This approximation uses as parameters the rgb color specifications for cyan ink, magenta ink, yellow ink, magenta+yellow ink together (red), cyan+yellow ink together (green), and cyan+magenta ink together (blue). These colors are specified in the array argument such that the rgb equivalent of Cyan ink is given by *(Cr, Cg, Cb)*, etc. *gamma* allows for compensation for nonlinearities in ink coverage; i.e. a cyan tint of value *c* is assumed to cover a fraction *c^gamma* of the paper.

The default values for these parameters are equivalent to:

```
[
    0.00784314    0.309804    1.0
    1.0           0.0         0.160784
    1.0           0.811765    0.0
    1.0           0.0         0.0
    0.0           0.294118    0.0156863
    0.0901961     0.0         0.164706
] 0.5 setcmykadjustparams
```

which provides a reasonable approximation for the inks used in the NeXT Color Printer as well as the C.E.I. standard inks.

These parameters are global to the PostScript interpreter, shared by all contexts.

currentcmykadjustparams *array* **currentcmykadjustparams** *array gamma*

Returns the current parameters for the Neugebauer approximation used by **setcmykadjust.** *array* must be of length 18, and is filled with the rgb components of the six ink colors (see **setcmykadjustparams**).

setdefaultcmykadjust *boolean* **setdefaultcmykadjust** -

Sets the default value of cmykadjust for the context in which it is issued. When set to *true*, all windows will default to having cmyk adjustment active. The default value of this default is *true* for new contexts.

currentdefaultcmykadjust - currentdefaultcmykadjust *boolean*

Returns the current default setting of cmykadjust for the context.

Bugs Fixed in Release 3.1

These bugs have been fixed in Release 3.1:

Reference	30925
Problem	Disk space leak when aborting print jobs.
Description	When print jobs prematurely aborted (due to an error, for example), PostScript was not cleaning up all references to the spooled data.
Reference	30866
Problem	Readimage on a small color window returns garbage alpha.

Description Pixel values from screen are now correctly stuffed with opaque alpha.

Reference 30784

Problem Certain Level 2 patterns can crash the WindowServer.

Description Level 2 patterns that render 2bit image data in their **/PaintProc** procedure would sometimes cause DPS to crash .

Reference 30695

Problem Type 2 imaging problems.

Description In certain cases, type 2 (i.e. window-to-window) imaging from a 2bit source to a 2bit partially-obscured destination would cause rendering errors.

Reference 30661

Problem Too many patterns.

Description Heavy pattern usage could cause the pattern caching machinery to get confused. The net result would be that drawing with `older' patterns (i.e. patterns least recently created) would have no effect.

Reference 30564

Problem	Grayscale image rendering on NeXT Laser Printer.
Description	Rendering certain 8bit images for printing on the 400dpi NeXT Laser Printer would crash the WindowServer.
Reference	30447
Problem	WindowServer doesn't timeshare User and Printer well.
Description	PostScript contexts involved in printing are now completely starved from execution as long as another context is runnable. This results in improved responsiveness of the UI during printing.
Reference	30146
Problem	2bit imaging problems.
Description	When imaging a 2 bit image into a 2 bit window with alpha, where the current alpha is not one, the resulting image would be corrupted.
Reference	29472
Problem	NXImageBitmap not clipped properly.
Description	If an NXImageBitmap was to be completely clipped, then no clipping was being done.

Reference	28901
Problem	Crash when printing a dvips output file.
Description	Certain Tek generated dvips files would cause the WindowServer to crash.
Reference	22307
Problem	Image don't respect the Decode array.
Description	Many cases of Level 2 imaging did not respect the `Decode' array entry in the image dictionary.
Reference	22021
Problem	Documents print as black rectangles.
Description	Occasionally documents would print as black rectangles on the NeXT Laser Printer.

Notes Specific to Release 3.0

These notes were included with the Release 3.0 version of the Window Server.

Sections that are no longer relevant have been marked with an italicized comment.

Major New Features in 3.0

The following new features have been added to the Window Server since Release 2.1.

- PostScript Level 2

  The NEXTSTEP 3.0 Window Server encorporates PostScript Level 2 from Adobe. This includes many extensions to the PostScript Language that have always been in Display PostScript, but also includes a number of new features such as

  · device independent color
  · patterns and forms
  · filters
  · setpagedevice

  For further information, consult the latest version of the PostScript Language Reference Manual (aka the *new* red book), published by Addison Wesley.

- Device Independent Color in NEXTSTEP 3.0

NEXTSTEP 3.0 contains enhancements to support device independent color. We have standardized on a calibrated RGB colorspace. All modes of the Application Kit's ColorPanel (except CMYK) now emit device independent colors. This means that applications using the Color Panel in conjuction with **NXSetColor()** will take full advantage Level 2 color when printing on PostScript Level 2 devices, including NeXT's new Color BubbleJet printer.

Device Independent Color Details

NXCalibratedRGB Color Space

The calibrated color space we have chosen matches the specifications of the standard NeXT MegaPixel Color Display. This color space is based on the CCIR Rec 709 phosphor set, balance to a white point of D65. This corresponds to CIE chromaticities of :

	x	y
Red	.640	.330
Green	.300	.600
Blue	.150	.060
White	.3127	.3290

Because **NXCalibratedRGB** is closely related to colorimetry of NeXT monitors, chosing this color space gains the advantage of calibrated color,

while sacrificing little if any performance for interactive rendering on the display.

The kit function **NXSetColor**() now uses a new operator **nxsetrgbcolor** to set its color when in any modes other than CMYK. **nxsetrgbcolor** and **nxsetgray** are defined as:

```
/nxsetrgbcolor { NXCalibratedRGBColorSpace setcolorspace setcolor } bind def
/nxsetgray { NXCalibratedRGBColorSpace setcolorspace dup dup setcolor } bind def
```

Note because the color space in Level 2 is implicitly changed by any of the Level I color operators (i.e. setrgbcolor, setgray) **setcolorspace** must be called explicitly each time the color is set.

- Spot Color

Spot color in PostScript refers to the color of fills, strokes, and characters (everything except sampled images). In NEXTSTEP documents, these attributes are typically controlled via **NXColor**s and the Color Panel. Because of the changes detailed above, these documents will now automatically inherit calibrated color.

Spot Color can also be calibrated through the use of the Pantone Color support in 3.0. Pantone colors have the advantage working on Level 1 devices as well.

· Sampled Images

Sampled images in NEXTSTEP are typically stored in TIFF files and drawn by the **NXBitmapImageRep** object. RGB TIFF files will be assumed to be in the **NXCalibratedRGBColorSpace** color space. To insure interactive performance to the screen images are rendered to the screen in **DeviceRGB** colorspace and to the printer in **CalibratedRGB** by default. To render images to the screen using a calibrated color space, the application writer is responsible for setting up the current colorspace using **`NXDrawBitmap()`** with the **`NX_CustomColorSpace`** option. Currently **NXBitmapImageRep** does not asupport the TIFF WhitePoint, PrimaryChromaticies, Gamma, and ColorResponseCurves, although this will occur in future releases of NEXTSTEP.

· Color Rendering Dictionaries:

The Color Rendering dictionary **NXCalibratedRGB** is pre-defined in **globaldict**. This rendering dictionary defines the corresponding color rendering for the **NXCalibratedRGB** color space. The concatination of the **NXCalibratedRGB** color space and the **NXCalibratedRGB** color rendering dictionary results in the identity transformation. This allows the PostScript to

optimize its calculation so that rendering with **NXCalibratedRGB** to the screen is as fast as rendering **DeviceRGB.**

· Resources

PostScript Level 2 supports a generalized resource scheme by which PostScript programs can refer to resources by name, as opposed to containing an entire definition of the resource. This is a generalization of the **definefont/findfont** mechanism. For full details, see the RedBook2. Resources exist on the *%resource%* device. On the NeXT, we search in three places for resources: ~/Library/PS2Resources/*category*, /LocalLibrary/PS2Resources/*category*, and ~/NextLibrary/PS2Resources/*category*. *category* is one of the regular or implicit resource categories defined in the RedBook2 on page 88. The following resources are installed in /NextLibrary/PS2Resources:

ColorRendering/NXCalibratedRGB
ColorSpace/NXCalibratedRGB
Patterns/NX_MediumGray
Patterns/NX_DarkGray
Patterns/NX_LightGray

Many of the predefined names which use these resources do so in a lazy

manner. For example, the definition of **NX_MediumGrayPattern** is

```
globaldict /NX_MediumGrayPattern {
        /NX_MediumGray /Pattern findresource
        matrix makepattern dup /NX_MediumGrayPattern exch store
} put
```

This loads the resource into VM upon first reference, and then rebinds the name to the actual instantiation of the pattern resource.

You can search for available resources with the *%resource%* device. For example,

```
(%resource%Pattern/*) {==} 1024 string filenameforall
```

prints a list of all the files which define patterns. Of course, you can also do

```
(*) {==} 1024 string /Pattern resourceforall
```

Level 2 Caveats:

This section attempts to clarify some of the interaction between NeXT's DPS extensions and the new features in Level 2.

- Alpha (transparency) is possible in Level 2 patterns, however these patterns are not portable to other PostScript implementations and the performance may not be as fast as if the pattern were opaque.

- It is not currently possible to use alpha in a Level 2 indexed color space.

- the window promotion logic has been extended in the following way to incorporate new level 2 features: Colored patterns promote windows as though the PaintProc were merely executed. Images in the CIEBasedA or CIEBasedABC colorspace (regardless of bit-depth) will promote the window as though an eight-bit per channel DeviceGray or DeviceRGB (respectively) image were drawn. Any image drawn in an Indexed colorspace will promote the window as though an eight-bit per channel DeviceGray or DeviceRGB image were drawn depending on the number of components in the underlying colorspace.

- since windowdevices can either be color or monochrome depending on their bit-depth, the semantics of color-rendering dictionaries on monochrome windows is changed slightly to include an implicit NTSC luminance sum (*gray* = .3**red* + .59**green* + .11**blue*) as the last step. This means that the same three-channel color rendering rendering dictionary (by default **NXCalibratedRGBRendering**) can be used for a window regardless of its promotion state. Monochrome color rendering dictionaries for printer devices behave as documented in the Red Book.

Changes

The following changes have been made to the Window Server since Release 2.1.

- Windows mistakenly promoted to 24 bits per pixel when rendering a 12 bit RGB image. This has been fixed to correctly promote only to 12 bits per pixel.

- Performing a PlusD **compositerect** in an 8 bit window didn't work properly. This has been fixed.

- On NeXTdimension, any color image being rendered into a 2 bit window didn't work. This has been fixed. (This was rarely seen unless the window was explicitly limited to 2-bits).

- The **image** operator has been optimized in the case of rectilinear scaling, with the addition of special case scaling code when the source and destination are the same pixel depth and no transfer function is in effect.

- **readimage** is no longer the most efficient way to read the bits **from** a window into an application. A new optimized pathway is available using the AppKit function **NXReadBitmap(),** (or, preferably, the **NXBitmapImageRep** object) which uses efficient Mach out-of-band messaging. **readimage** will continue to be support for backwards compatibility.

- out-of-band images which are passed **into** the window server (rendered via **NXDrawBitmap()**, or the **NXImageBitmapRep** object) can now have arbitrary **`rowBytes'**. This means that the pixels to be imaged need not be tightly

packed into scanlines. This allows sub-regions of a bitmap stored on the client side to be passed to the window server without reformatting.

- out-of-band images passed **into** the window server are now allowed to pack non-alpha image data on natural machine-word boundaries. In particular 12 bit RGB data can be passed in 16 bit pixels (the alpha nibble MUST BE SET to 0xF). Similarly, 24bit non-alpha data can be passed in 32 bit RGB pixels (the alpha byte MUST BE SET to 0xFF). This allows applications to store rasters in a more natural way for their own processing, and minimizes the need for reformatting the data to pass to the window server for display. This capability is available through the AppKit's **NXDrawBitmap()** function

- On the 68040 processor, we have improved the performance of the **image** operator in the case of unpacked 12 bit RGB data imaged (with the identity transform) into a 16 bit window. In order to take advantage of this performance improvement, your source data must be short-aligned (guaranteed if you malloc your source buffer) and its rowbytes must be a multiple of 4 (you can easily insure this if you are generating the source data and rowbytes yourself). In this case, a 400x400 image should take 55-60ms on a 25Mhz NeXTstation Color.

- There is an additional performance gain above and beyond that mentioned above for an even more strict set of alignment restrictions: Your source data pointer is cacheline-aligned (16 byte aligned) with the destination data pointer that it will be written into and your source rowbytes modulo 16 equals the

destination buffer rowbytes modulo16. Since your application cannot know the data pointer alignment of a buffered window, this is very difficult when imaging into a buffered window. But, if you are imaging into a retained window, the destination data pointer modulo 16 equals the x coordinate on screen multiplied by 2, and the rowbytes of the destination == 0 modulo 16. So, if you image into a retained window and align the address of the beginning of your source buffer to match the twice x coordinate (modulo 16) that it will be imaged to on the screen, and you make its rowbytes a multiple of 16 (to match the 0 modulo 16), then your image data will be copied to the screen in the fastest way possible (using the move16 instruction newly available on the 040). For large images this is 45% faster than case (4). In this case, a 400x400 image should take 37-40ms on a 25Mhz NeXTstation Color.

- **setpagedevice** raises a **configurationerror** if the current device is not a page device or a window device. If the current device is a page device, then **setpagedevice** behaves as documented in the *PostScript Language Reference Manual, 2nd Edition*. If the current device is a window device, then the dictionary given to **setpagedevice** is discarded and **erasepage** followed by **initgraphics** is executed.

New (NeXT) PostScript Operators

The following PostScript operators have been added or modified since Release 2.0.

The operators marked *internal* shouldn't be used in applications based on the Application Kit since your use of them may conflict with the Kit's. For descriptions of these operators, see the *NEXTSTEP General Reference.* (Use the Digital Librarian to view the on-line version of the reference manual.)

image % New DPS extension
setshowpageprocedure % public
currentshowpageprocedure % public
setframebuffertransfer % public
settrackingrect % new features
currentframebuffertransfer % public

currentshowpageprocedure *win* **currentshowpageprocedure** *proc*

Returns the current showpage procedure for the window specified by *win*. The showpage procedure is a PostScript procedure which will be executed each time the **showpage** operator is executed when the currentdevice is the given window.

ERRORS
invalidid, **stackunderflow**, **typecheck**

SEE ALSO

setshowpageprocedure

currentframebuffertransfer *fbnum* **currentframebuffertransfer** *redproc greenproc blueproc grayproc*

Returns the current transfer functions in effect for the framebuffer indexed by *fbnum.* *fbnum* ranges from 0 to **countframebuffers** - 1.

ERRORS
invalidid, **stackunderflow**, **typecheck**

SEE ALSO
setframebuffertransfer, countframebuffers, framebuffer

setframebuffertransfer *redproc greenproc blueproc grayproc* *fbnum* **setframebuffertransfer**

Sets the framebuffer transfer functions in effect for the framebuffer indexed by *fbnum.* *fbnum* ranges from 0 to **countframebuffers** - 1. The framebuffer transfer describes the relationship between the framebuffer values of the display, and the voltage produced to drive the monitor. **Important**: these transfer functions are

outside of the domain of PostScript Level 2's imaging model. NeXT's WindowServer assumes that the framebuffer values are directly proportional to screen brightness. This is important for the accuracy of dithering, compositing, and similar calculations. This API is provided only for the purpose of developers developing screen-calibration products. The default transfer for NeXT Color Displays is "{ 1 2.2 div exp } bind dup dup {}". The procedures must be allocated in shared VM.

It is possible to make framebuffer transfer functions persist beyond the lifetime of the WindowServer by storing a property in the NetInfo `screens' database. In the local NetInfo domain, /localconfig/screens holds the configuration information for the screens known to the WindowServer (i.e. MegaPixel, NeXTdimension). These specify the layout and activation state of the screen. A new property for 3.0 has been added called `defaultTransfer'. This property can contain a string of PostScript code suitable for execution by the **setframebuffertransfer** operator (without the *fbnum* parameter). For example the following represents the NetInfo configuration for a NeXTdimension screen with a default gamma of 2.0.

```
localhost:1#  niutil -read  .
```

```
/localconfig/screens/NeXTdimension
name: NeXTdimension
slot: 2
unit: 0
defaultTransfer: {1 2.0 div exp } dup dup dup
bounds: 0 1120 0 832
active: 1
_writers: *
```

The NetInfo `defaultTransfer' property is used to configure the screen each time the WindowServer starts up. This allows monitor calibration products to save their settings so the next time the WindowServer starts up, the new values will be used. Note that in some cases, the NetInfo configuration state for a monitor will not have active == 1, although the monitor is being used by the WindowServer. This is because if no screens are marked `active' in NetInfo, the WindowServer uses a suitable default. If a screen is not marked `active' in NetInfo, the other properties (such as `defaultTransfer') are ignored. So be sure that the screen for which you are adding a `defaultTransfer'entry is marked `active'.

Monochrome devices will ignore *redproc, greenproc,* and *blueproc*, RGB devices will ignore *grayproc.*

setframebuffertransfer is unsupported on the current

generation of NeXT monochrome displays.

ERRORS
invalidid, **stackunderflow**, **typecheck**

SEE ALSO
setframebuffertransfer, countframebuffers, framebuffer

setshowpageprocedure *proc win* **setshowpageprocedure** -

Sets the showpage procedure for the window specified by *win*. The showpage procedure is a PostScript procedure which will be executed each time the **showpage** operator is executed when the device in the current graphics state is the given window. *proc* must be allocated in shared VM.

ERRORS
invalidid, **stackunderflow**, **typecheck**

SEE ALSO
currentshowpageprocedure

settrackingrect *x y width height optionarray trectnum gstate* **settrackingrect ±**

In this alternate form, **settrackingrect**, can be used to set special event-gathering attributes of a rectangle. In this case events are not generated when the boundary is crossed, but certain low-level event system options can be controlled in the rectangle. *optionarray* contains key-value pairs that can, for example, affect pressure sensitivity in a rectangle.

This is an extensible API allowing the addition of new options as they are needed. Eventually the old tracking rect paramters will be subsumed by this new form of the operator. An empty option array is meaningless and will raise a **rangecheck** error. This form of the operator does not currently post mouse-entered and mouse-exited events. The following key-value pairs for *optionarray* are initially defined:

Key	Type	Semantics
Pressure	bool	treat any non-zero pressure values as a mouse-down. (false is default)
Coalesce	bool	coalesce mouse motion events (true is default)

Use the existing **cleartrackingrect** operator to remove special tracking rects. An example which turns pressure on and coalescing off would be:

```
x y width height [/Pressure true /Coalesce false]
trectnum gstate settrackingrect
```

ERRORS
invalidid, **stackunderflow**, **typecheck**

SEE ALSO

image *dict* **image** **±**

The Level 2 image operator has been extended for Display PostScript Systems only. The changes allow the image operator to use a window as source sample data. There are several reasons why an application may want to do this. For example, an application for viewing large photographic images such as satellite images can use the image operator for panning and zooming. If the image data is placed in an off-screen window, panning can be accomplished by copying the data to the on-screen window, and zooming can be accomplished by copying the data with a scale. In this example, pixel expansion is desirable, so that all of the image data is seen. Another

example in which a scale in the transformation is useful, is with bitmap editors. The pixels can be magnified for editing, then changed in the source sample data. Finally there is the advantage of using image data that is already accessible to the server. This improves performance since the image data does not need to be transferred from the client to the server. It is especially important if the server executes on a different system or processor than the client.

Image Dictionary Changes

Below is a description of the image dictionary for when windows are used as sources. In this case ImageType has a value of 2, and only the keys described in the table below have significance.

The **DataSource** entry can now be a graphics state object. The graphics state object contains a reference to the destination device. It is used as a device independent representation of a window. The pixel representation of the source and the color space for the image is determined by the device in the source graphics state object. The only information used in the source graphics state is the device structure and the CTM. No

assumptions about how the source was created will be made by the interpreter. Therefore it is up to the user to make sure that the desired effects will happen with the current transfer function and halftone screen. If the source gstate is the current gstate, and the source rectangle overlaps the destination, then the image will be rendered as if all of the source pixels had been read before imaging to the destination.

The **UnpaintedPath** entry is used when the pixels in a window were not available because the source rectangle includes areas outside of the clipping area for the window. This region is defined by the intersection of the source gstate's clip, the source gstate's viewclip, and the underlying window system's clip for the window. The **UnpaintedPath** will contain a userpath that encloses the entire area that was not available in the source. The userpath may be larger than the actual area that needs to be repainted, therefore a clip based on the source rectangle should be used. This clip, in addition to the **UnpaintedPath** can be used a clip for redrawing the PostScript code that created the source window, into the destination. Applications need to consider if a scale was used because pixel expansion or image interpolation can occur. In those cases, the missing area needs to be

rendered off-screen and then reimaged into the missing area. In most cases, an application will not have to worry about handling missing pixels, because this can be taken care of in a tool kit. The **UnpaintedPath** entry will be filled in by the image operator only if the key exists in the image dictionary that is passed to image. Otherwise, the unpainted area will not be calculated and returned.

PixelCopy is used when the application does not want the color conversion, transfer function or halftone screen to have an effect on the image operation. This is useful when the source window was generated by another imaging model besides PostScript. It's color space may not be part of the PostScript imaging model, but the data can still be copied and transformed.

Key	Type	Semantics
ImageType	integer	(*Required*) Must be 2 for gstate object sources.
XOrigin	real	(*Required*) X origin of source rectangle in user space coordinates as specified by the transformation in the

		DataSource graphics state object.
YOrigin	real	(*Required*) Y origin of source rectangle in user space coordinates as specified by the transformation in the **DataSource** graphics state object.
Width	real	(*Required*) Width of the rectangle to be copied in user space coordinates as specified by the transformation in the **DataSource** graphics state object.
Height	real	(*Required*) Height of the rectangle to be copied in user space coordinates as specified by the transformation in the **DataSource** graphics state object.
ImageMatrix	array	no change

DataSource	gstate	(*Required*) A gstate object that contains the device that will be used for the source sample data. This device will also be used to determine the pixel representation for the source, and the color space to be used by image.
Interpolate	boolean	no change
UnpaintedPath		various (*Returned Value*) If some of the pixels in the source were not available (because of clipping), then the **UnpaintedPath** entry contains a userpath in the current (destination) user space coordinates, that encloses the area that could not be filled. If all of the source pixels could be accessed, then the **UnpaintedPath** entry will contain a null object.

PixelCopy	boolean	(*Optional*) If true, indicates that the source pixels should be copied directly, without going through the normal color conversion, transfer function, or halftoning. The number of bits per pixel of the source must match the number of bits per pixel of the destination, otherwise a **typecheck** error will occur. If false or not present, the pixels will be imaged in the usual way.

Using image with gstate sources is similar to using image with any other sample data. One difference that should be considered is the source gstate's transformation matrix. Since the source rectangle is specified in the source's user space coordinates, the source gstate's CTM is included in the transformation. Therefore the mapping from the current user space to the image space is defined by the concatenation of the image matrix and the source gstate's CTM.

ERRORS

invalidid, **rangecheck**, **stackunderflow**, **typecheck**

SEE ALSO

alphaimage, **colorimage**, **image (red book)**