8 Filter Modules

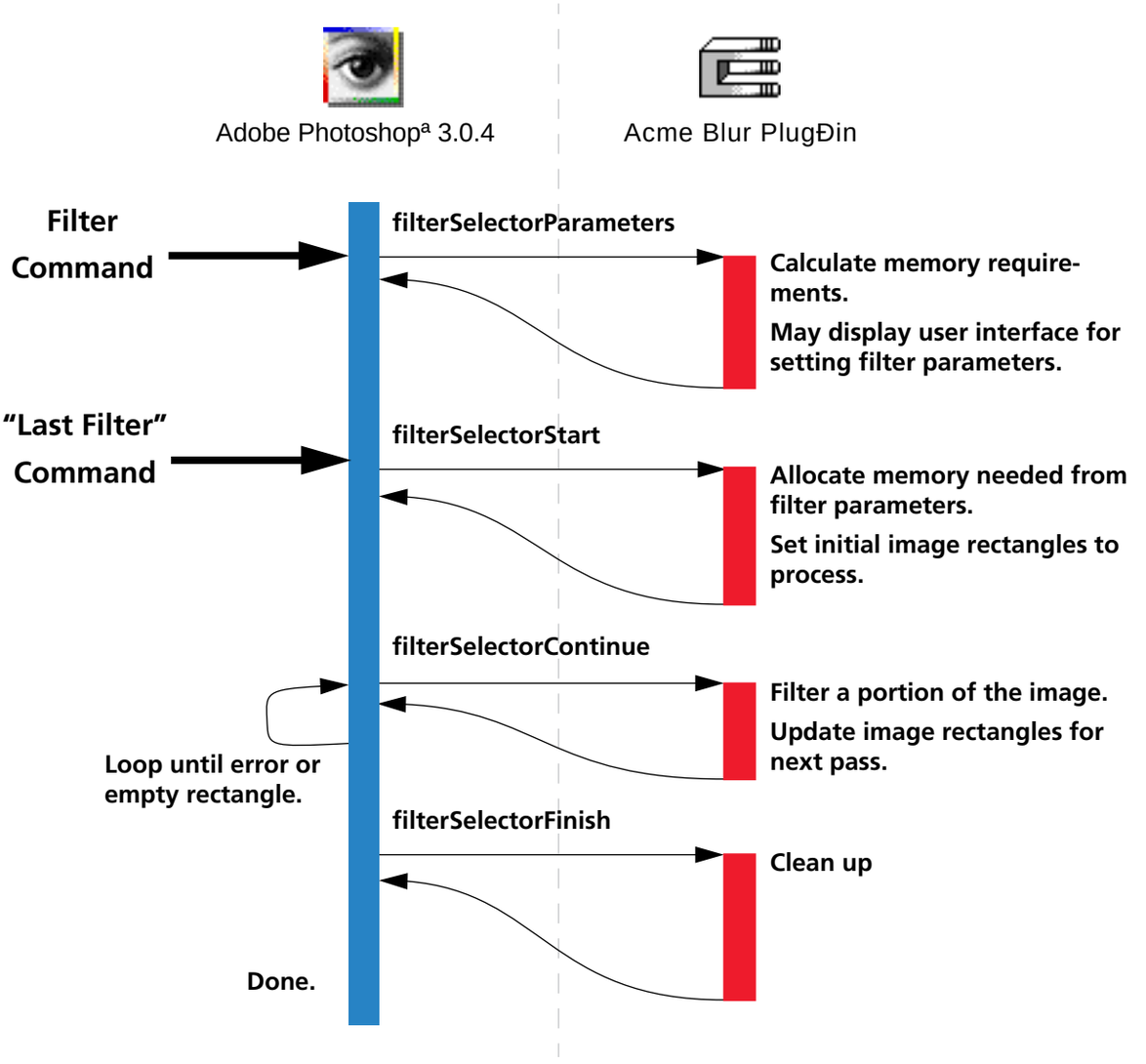
Filter modules modify a selected area of an image, and are accessed under the Filter menu. Filter actions range from subtle shifts of hue or brightness, to wild changes that create stunning visual effects.

Under Mac OS, the code resource and file type for filter modules is '8BFM'. Under Windows, the file extension is .8BF.

Sample plug–in

Dissolve is a sample filter plug–in which also demonstrates how to manipulate Photoshop's layers.

Calling sequence



When the user invokes a filter plug–in module by selecting its name from the Filter menu, Photoshop calls it with the sequence of selector values shown in the figure above. The actions for these selectors is discussed next.

filterSelectorParameters

If the plug–in filter has any parameters that the user can set, it should prompt the user and save the parameters in a relocatable memory block whose handle is stored in the parameters field. Photoshop initializes the parameters field to NULL when starting up.

This routine may or may not be called depending on how the user invokes the filter. After a filter has been invoked once, the user may re–apply that same filter with the same parameters (this is the "Last Filter" command under the Filter menu). In this case, the plug–in host does not call filterSelectorParameters, and the user will not be shown any dialogs to enter new parameters.

Since the same parameters can be used on different size images, the parameters should not depend on the size or mode of the image, or the size of the filtered area (these fields are not even defined at this point).

A future version of Photoshop may have a macro processor which would save the block pointed to by the parameters field when recording, so that it can operate the filter without user input during play back (Adobe Premiere™ actually uses this feature). To be compatible with this feature, all parameters

must be saved in a relocatable block whose handle is stored in the parameters field.

The parameter block should contain the following information:

1. A signature so that the plug–in can do a quick confirmation that this is, in fact, one of its parameter blocks.

2. A version number so that the plug–in can evolve without requiring a new signature.

3. A convention regarding byte–order for cross–platform support (or a flag to indicate what byte order is being used).

The plug–in should validate the contents of its parameter handle when it starts processing if there is a danger of it crashing from bad parameters.

You may wish to design your filter module so that you store default values or the last used set of values for your parameter block in the filter module's resource fork (Mac OS) or another file (Windows). This way, you can save preference settings for your filter plug–in across invocations of the plug–in host.

filterSelectorPrepare

The filterSelectorPrepare selector calls allow your plug–in module to adjust Photoshop's memory allocation algorithm. Photoshop sets **maxData** to the maximum number of bytes it can allocate to your plug–in. You may want to reduce **maxData** for increased efficiency. Refer to chapter 3 for details on memory management strategies.

The fields such as **imageSize**, **planes**, **filterRect**, etc. have now been defined, and can be used in computing your buffer size requirements.

If your plug–in filter module is planning on allocating any large (>= about 32K) buffers or tables, you should set the **bufferSpace** field to the number of bytes you are planning to allocate. Photoshop will then try to free up that amount of memory before calling the plug–in's filterSelectorStart handler.

Alternatively, you can set this field to zero and use the buffer and handle suites if they are available.

filterSelectorStart

Your plug–in should set **inRect** and **outRect** (and **maskRect**, if it is using the selection mask) to request the first areas of the image to work on.

If at all possible, you should process the image in pieces to minimize memory requirements. Unless there is a lot of startup/shutdown overhead on each call (for example, communicating with an external DSP), tiling the image with rectangles measuring 64x64 to 128x128 seems to work fairly well.

filterSelectorContinue

Your filterSelectorContinue handler is called repeatedly as long as at least one of the **inRect**, **outRect**, or **maskRect** fields is non–empty.

Your handler should process the data pointed by **inData** and **outData** (and possibly **maskData**) and then update **inRect** and **outRect** (and **maskRect**, if using the selection mask) to request the next area of the image to process.

filterSelectorFinish

This call allows the plug–in to clean up after a filtering operation. This call is made if and only if the filterSelectorStart handler returns without error, even if the filterSelectorContinue routine returns an error.

If Photoshop detects Command–period (Mac OS) or Escape (Windows) between calls to the filterSelectorContinue routine, it will call the filterSelectorFinish routine. Be careful here, since normally the plug–in would be expecting another filterSelectorContinue call.

Notes:

1. If filterSelectorStart succeeds, then Photoshop guarantees that filterSelectorFinish will be called.

2. Photoshop may choose to go to filterSelectorFinish instead of filterSelectorContinue if it detects a need to terminate while fulfilling a request.

3. AdvanceState may be called from either filterSelectorStart or filterSelectorContinue and will drive Photoshop through the buffer set up code. If the rectangles are empty, the buffers will simply be cleared. Termination is reported as userCanceledErr in the result from the advanceState call.

Error return values

The plug–in module may return standard operating system error codes, or report its own errors, in which case it can return any positive integer.

```
#define filterBadParameters  –30100  // a problem with the interface
#define filterBadMode        –30101  // module doesn’t support <mode> images
```

The Filter parameter block

The pluginParamBlock parameter passed to your plug–in module's entry point contains a pointer to a FilterRecord structure with the following fields. This structure is declared in PIFilter.h.

Table 8–1: FilterRecord structure

Type	Field	Description
int32	serialNumber	This field contains Adobe Photoshop's serial number. Your plug–in module can use this value for copy protection, if desired.
TestAbortProc	abortProc	This field contains a pointer to the TestAbort callback documented in chapter 3.
ProgressProc	progressProc	This field contains a pointer to the UpdateProgress callback documented in chapter 3. This procedure should only be called during the actual main operation of the plug–in, not during long operations during the preliminary user interface such as building a preview.
Handle	parameters	Photoshop initializes this handle to NULL at startup. If your plug–in filter has any parameters that the user can set, you should allocate a relocatable block in your filterSelectorParameters handler, store the parameters in the block, and store the block's handle in this field.
Point	imageSize	The image's width (imageSize.h) and height (imageSize.v) in pixels. If the selection is floating, this field instead holds the size of the floating selection.
int16	planes	For version 4 filters, this field contains the total number of active planes in the image, including alpha channels. The image mode should be determined by looking at **imageMode**. For pre–version 4 filters, this field will be equal to 3 if filtering the RGB "channel" of an RGB color image, or 4 if filtering the CMYK "channel" of a CMYK color image. Otherwise it will be equal to 1.
Rect	filterRect	The area of the image to be filtered. This is the bounding box of the selection, or if there is no selection, the bounding box of the image. If the selection is not a perfect rectangle, Photoshop automatically masks the changes to the area actually selected (unless the plug–in turns off this feature using autoMask). This allows most filters to ignore the selection mask, and still operate correctly.
RGBColor	background	The current background and foreground colors. If planes is equal to 1, these will have already been converted to monochrome. (Obsolete: Use backColor and foreColor.)
RGBColor	foreground	

Table 8–1: FilterRecord structure (Continued)

Type	Field	Description
int32	maxSpace	This lets the plug–in know the maximum number of bytes of information it can expect to be able to access at once (input area size + output area size + mask area size + bufferSpace).
int32	bufferSpace	If the plug–in is planning on allocating any large internal buffers or tables, it should set this field during the filterSelectorPrepare call to the number of bytes it is planning to allocate. Photoshop will then try to free up the requested amount of space before calling the filterSelectorStart routine.
Rect	inRect	Set this field in your filterSelectorStart and filterSelectorContinue handlers to request access to an area of the input image. The area requested must be a subset of the image's bounding rectangle. After the entire filterRect has been filtered, this field should be set to an empty rectangle.
int16	inLoPlane	Your filterSelectorStart and filterSelectorContinue handlers should set these fields to the first and last input planes to process next.
int16	inHiPlane	
Rect	outRect	Your plug–in should set this field in its filterSelectorStart and filterSelectorContinue handlers to request access to an area of the output image. The area requested must be a subset of **filterRect**. After the entire **filterRect** has been filtered, this field should be set to an empty rectangle.
int16	outLoPlane	Your filterSelectorStart and filterSelectorContinue handlers should set these fields to the first and last output planes to process next.
int16	outHiPlane	
void *	inData	This field contains a pointer to the requested input image data. If more than one plane has been requested (**inLoPlane** is not equal to **inHiPlane**), the data is interleaved.
int32	inRowBytes	The offset between rows of the input image data. There may or may not be pad bytes at the end of each row.
void *	outData	This field contains a pointer to the requested output image data. If more than one plane has been requested (**outLoPlane** is not equal to **outHiPlane**), the data is interleaved.
int32	outRowBytes	The offset between rows of the output image data. There may or may not be pad bytes at the end of each row.
Boolean	isFloating	This field is set TRUE if and only if the selection is floating.
Boolean	haveMask	This field is set TRUE if and only if a non–rectangular area has been selected.

Table 8–1: FilterRecord structure (Continued)

Type	Field	Description
Boolean	autoMask	By default, Photoshop automatically masks any changes to the area actually selected. If **isFloating** is FALSE, and **haveMask** is TRUE, your plug–in can turn off this feature by setting this field to FALSE. It can then perform its own masking.
Rect	maskRect	If **haveMask** is TRUE, and your plug–in needs access to the selection mask, your should set this field in your filterSelectorStart and filterSelectorContinue handlers to request access to an area of the selection mask. The requested area must be a subset of **filterRect**. This field is ignored if there is no selection mask.
void *	maskData	A pointer to the requested mask data.
int32	maskRowBytes	The offset between rows of the mask data.
FilterColor	backColor	The current background and foreground colors, in the color space native to the image.
FilterColor	foreColor	
OSType	hostSig	The plug–in host provides its signature to your plug–in module in this field. Photoshop's signature is '8BIM'.
HostProc	hostProc	If not NULL, this field contains a pointer to a host–defined callback procedure that can do anything the host wishes. Plug–ins should verify **hostSig** before calling this procedure. This provides a mechanism for hosts to extend the plug–in interface to support application specific features.
int16	imageMode	The mode of the image being filtered (Gray Scale, RGB Color, etc.). See the header file for possible values. Your filterSelectorStart handler should return a filterBadMode result code if it is unable to process this mode of image.
Fixed	imageHRes	The image's horizontal and vertical resolution in terms of pixels per inch. These are fixed point numbers (16 binary digits).
Fixed	imageVRes	
Point	floatCoord	If isFloating is TRUE, the coordinate of the top–left corner of the floating selection in the main image's coordinate space.
Point	wholeSize	If isFloating is TRUE, the size in pixels of the entire main image.
PlugInMonitor	monitor	This field contains the monitor setup information for the host. See appendix A for details.
void *	platformData	This field contains a pointer to platform specific data. Not used under Mac OS.
BufferProcs *	bufferProcs	This field contains a pointer to the buffer suite if it is supported by the host, otherwise NULL. See chapter 3 for details.

Table 8–1: FilterRecord structure (Continued)

Type	Field	Description
ResourceProcs *	resourceProcs	This field contains a pointer to the pseudo–resource suite if it is supported by the host, otherwise NULL. See chapter 3 for details.
ProcessEventProc	processEvent	This field contains a pointer to the ProcessEvent callback. It contains NULL if the callback is not supported. See chapter 3 for details.
DisplayPixelsProc	displayPixels	This field contains a pointer to the DisplayPixels callback. It contains NULL if the callback is not supported. See chapter 3 for details.
HandleProcs *	handleProcs	This field contains a pointer to the handle callback suite if it is supported by the host, otherwise NULL. See chapter 3 for details.
These fields are new in version 3.0.		
Boolean	supportsDummyPlanes	Does the host support the plug–in requesting non–existent planes? (see dummy planes fields, below) This field is set by the host to indicate whether it respects the dummy planes fields.
Boolean	supportsAlternateLayouts	Does the host support data layouts other than rows of columns of planes? This field is set by the plug–in host to indicate whether it respects the wantLayout field.
int16	wantLayout	The desired layout for the data. See PIGeneral.h. The plug–in host only looks at this field if it has also set **supportsAlternateLayouts**.
int16	filterCase	The type of data being filtered, flat, floating, layer with editable transparency, layer with preserved transparency. With and without a selection. A zero indicates that the host did not set this field.
int16	dummyPlaneValue	The value to store into any dummy planes. 0..255 = specific value. –1 = leave undefined (i.e., random)
void *	premiereHook	See the Adobe Premiere Plug–in Developer's Kit.
AdvanceStateProc	advanceState	The AdvanceState callback. See chapter 3 for details on this callback function.
Boolean	supportsAbsolute	Does the host support absolute channel indexing? Absolute channel indexing ignores visiblity concerns and numbers the channels from zero starting with the first composite channel if any, followed by the transparency, followed by any layer masks, followed by any alpha channels.
Boolean	wantsAbsolute	Enable absolute channel indexing for the input. This is only useful if **supportsAbsolute** is TRUE. Absolute indexing is useful for things like accessing alpha channels.

Table 8–1: FilterRecord structure (Continued)

Type	Field	Description
GetPropertyProc	getProperty	The GetProperty callback. This direct callback pointer has been superceded by the property callback suite, but is maintained here for backwards compatibility. See chapter 3 for details.
Boolean	cannotUndo	If the filter makes a non–undoable change, then setting this field will prevent Photoshop from offering undo for the filter. This is rarely needed.
int16	inputPadding	The input, output, and mask can be padded when loaded. The options for padding include specifying a specific value (0..255), specifying edge replication (plugInWantsEdgeReplication), specifying that the data be left random (plugInDoesNotWantPadding), or requesting that an error be signaled for an out of bounds request (plugInWantsErrorOnBoundsException). The error case is the default since previous versions would have errored out in this event.
int16	outputPadding	
int16	maskPadding	
char	samplingSupport	Does the host support non–1:1 sampling of the input and mask? Photoshop 3.0.1 supports integral sampling steps (it will round up to get there). This is indicated by the value hostSupportsIntegralSampling. Future versions may support non–integral sampling steps. This will be indicated with hostSupportsFractionalSampling.
char	reservedByte	(for alignment)
Fixed	inputRate	The sampling rate for the input. The effective input rectangle (in normal sampling coordinates) is **inRect** * **inputRate** (i.e., inRect.top * inputRate, inRect.left * inputRate, inRect.bottom * inputRate, inRect.right * inputRate). inputRate is rounded to the nearest integer in Photoshop 3.0.1. Since the scaled rectangle may exceed the real source data, it is a good idea to set some sort of padding for the input as well.
Fixed	maskRate	Like **inputRate**, but as applied to the mask data.
int16	inLayerPlanes	The number of planes (channels) in each category for the input data. This is the order in which the planes are presented to the plug–in and as such gives the structure of the input data. The inverted layer masks are ones where 0 = fully visible and 255 = completely hidden. If these are all zero, then the plug–in should assume the host has not set them.
int16	inTransparencyMask	
int16	inLayerMasks	
int16	inInvertedLayerMasks	
int16	inNonLayerPlanes	

Table 8–1: FilterRecord structure (Continued)

Type	Field	Description
int16	outLayerPlanes	The structure of the output data. This will be a prefix of the input planes. For example, in the protected transparency case, the input can contain a transparency mask and a layer mask while the output will contain just the layerPlanes.
int16	outTransparencyMask	
int16	outLayerMasks	
int16	outInvertedLayerMasks	
int16	outNonLayerPlanes	
int16	absLayerPlanes	The structure of the input data when **wantsAbsolute** is TRUE.
int16	absTransparencyMask	
int16	absLayerMasks	
int16	absInvertedLayerMasks	
int16	absNonLayerPlanes	
int16	inPreDummyPlanes	The number of extra planes before and after the input data. This is only available if **supportsDummyChannels** is TRUE. This is used for things like forcing RGB data to appear as RGBA.
int16	inPostDummyPlanes	
int16	outPreDummyPlanes	Like **inPreDummyPlanes** and **inPostDummyPlanes**, except it applies to the output data.
int16	outPostDummyPlanes	
int32	inColumnBytes	The step from column to column in the input. If using the layout options, this value may change from being equal to the number of planes. If it is zero, you should assume that the plug–in host has not set it.
int32	inPlaneBytes	The step from plane to plane in the input. Normally one, but this changes if the plug–in uses the layout options. If it is zero, you should assume that the plug–in host has not set it.
int32	outColumnBytes	The output equivalent of the previous two fields.
int32	outPlaneBytes	
These fields are new in version 3.0.4.		
ImageServicesProcs *	imageServicesProcs	This is a pointer to the image services callback suite. See chapter 3 for details.
int16	inTileHeight	Tiling for the input.
int16	inTileWidth	
Point	inTileOrigin	
int16	absTileHeight	Tiling for the absolute data.
int16	absTileWidth	
Point	absdTileOrigin	
int16	outTileHeight	Tiling for the output.
int16	outTileWidth	
Point	outTileOrigin	
int16	maskTileHeight	Tiling for the mask.
int16	maskTileWidth	
Point	maskTileOrigin	

9 Format Modules

Format plug–in modules (sometimes referred to as image format, or file format modules) are used to add new file types to the Open..., Save, and Save As... commands. Adobe Photoshop is shipped with several file format modules to read and write different formats including GIF, MacPaint, and BMP.

Acquire and export modules may also be used to read and write files. You should create a format module rather than an acquire and/or export module if you want your users to treat your files in the same fashion as native Photoshop files. In particular you should use a format module if:

- You want users to be able to create, modify, save, and re–open files in your format. If your format uses a lossy compression algorithm, you may want to consider image degradation issues for this situation.
- You want users to be able to double–click a document to launch Photoshop (Mac OS), or associate your file extension with the Photoshop application (Windows).

You may not want to use a format module if:

- With respect to Photoshop, your file format is read–only or write–only.
- The image compression and/or color space conversion necessary for your file format would result in unacceptable image degradation if users read and saved repeatedly.

Under Mac OS, the code resource and file type for format modules is '8BIF'. Under Windows, the file extension is .8BI.

Sample plug–in

Sample Format is a sample format module. This module is written to use the new AdvanceStateProc callback, introduced in Photoshop 3.0.

Format module operations

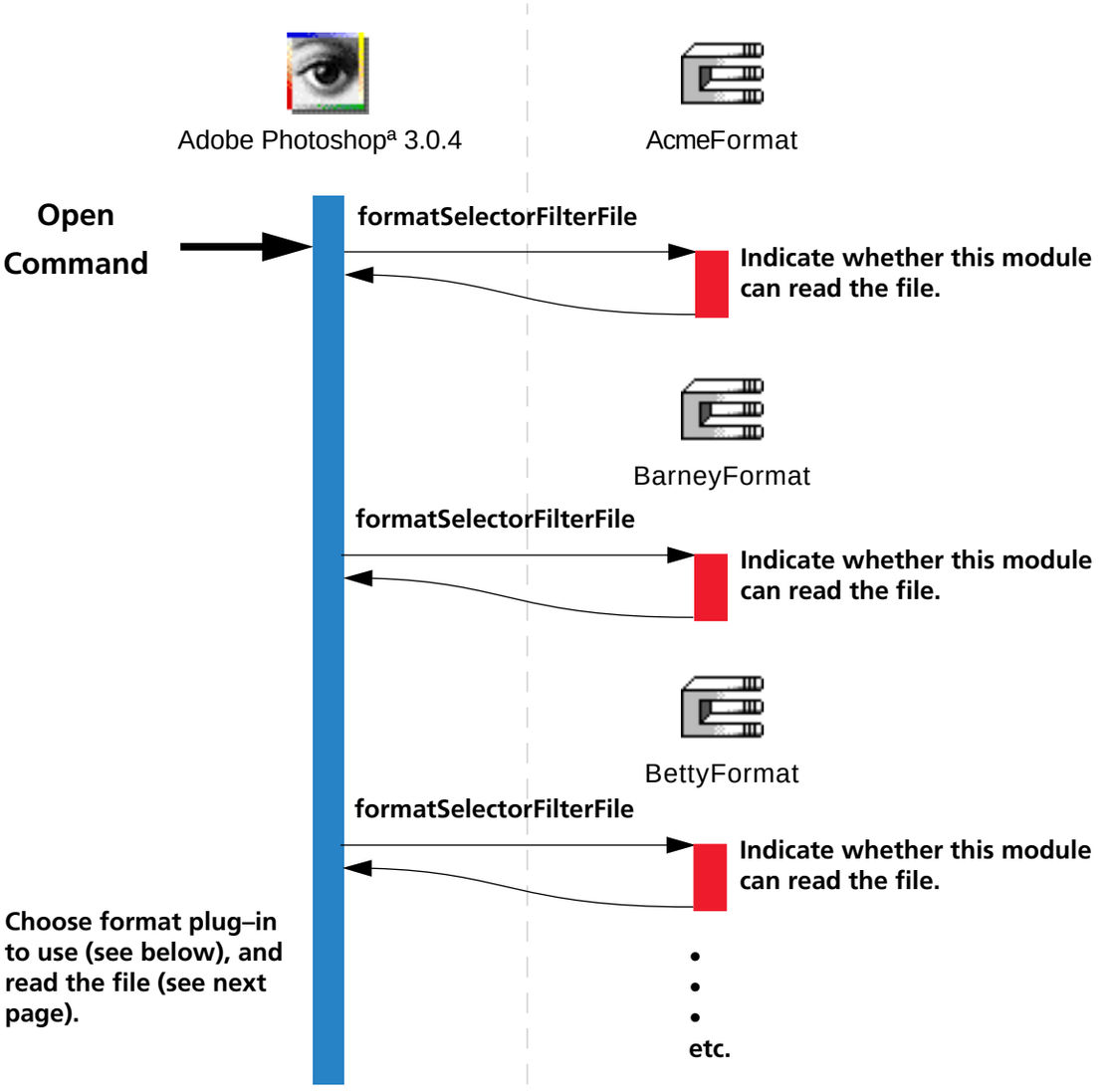
File format plug–in modules have two main functions: reading an image from a file, and writing an image to a file. Reading a file is a two step process:

- The **formatSelectorFilterFile** selector is used to determine whether a format module can read a particular file. This selector is called when the user performs an Open command, and is described in more detail on the next page.
- The **read sequence** is used to read image files.

Writing a file consists of three sequences:

- The **options sequence** is used to request save options from the user. It will only be used when first saving a document in a particular format.
- The **estimate sequence** estimates the file size so that the host can decide whether there is enough disk space available.
- The **write sequence** actually writes the file.

Reading a file (file filtering)



When the user selects a file with the Open... command from the file menu, there may be one or more format module that lists the file type (Mac OS) or file extension (Windows) as a supported format. For each such plug–in module, Photoshop will call the plug–in with a **formatSelectorFilterFile** selector. The plug–in module should then examine the file to determine whether the file is one that it can process, and indicate this in its **result** parameter:

```
if (module can read this file)
    *result = noErr;
else
    *result = formatCannotRead;
```

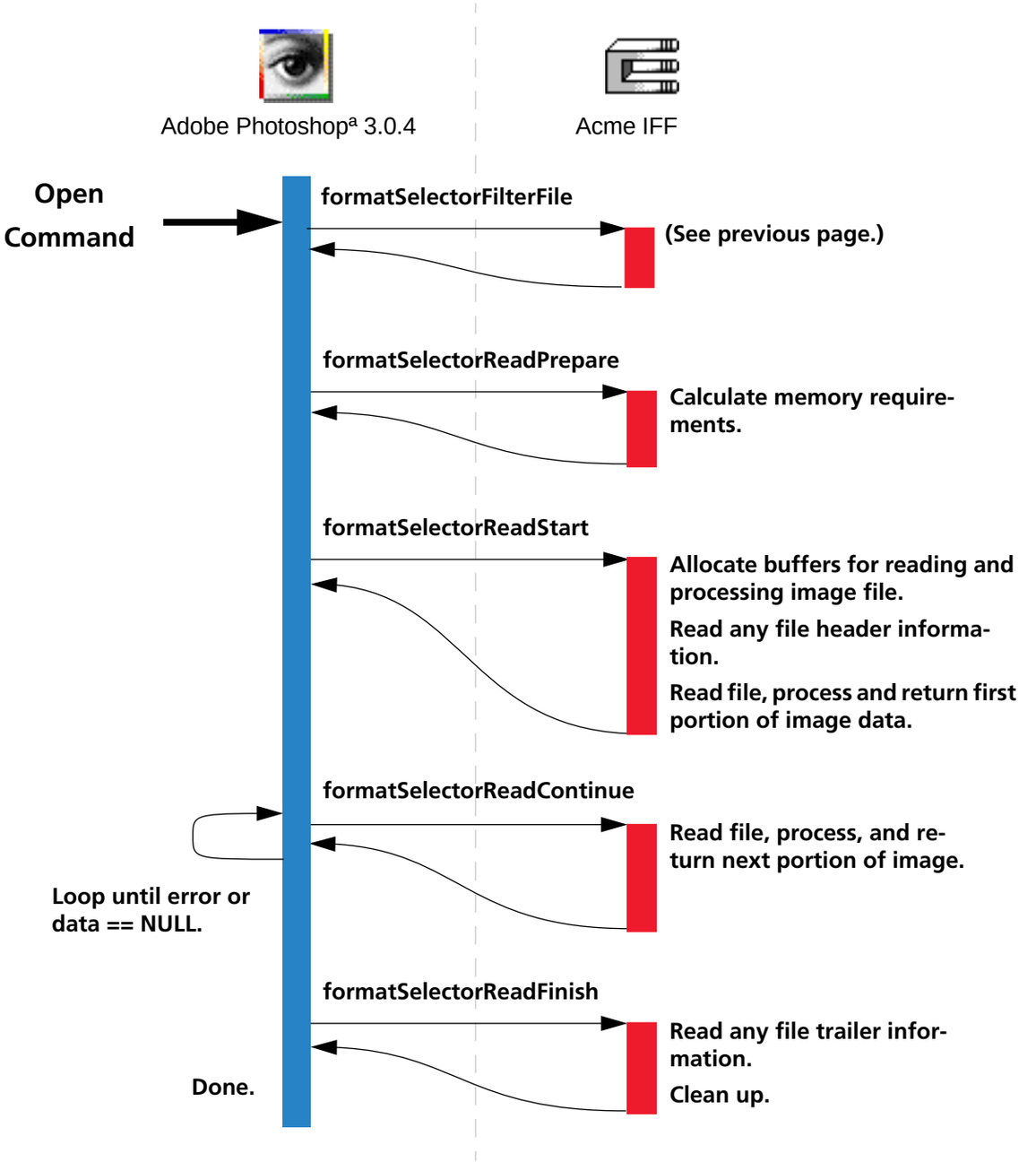
If more than one format module can read the file, Photoshop uses the following priority scheme to determine which plug–in module to use:

(1) The module with the first PICategoryProperty string (sorted alphabetically) will be used. Modules with no PICategoryProperty will default to their PINameProperty for this comparison.

(2) If two or more modules have matching category names, the module with the highest PIPriorityProperty value will be used.

(3) If two or more modules have matching category and priority, which module will be selected is undefined.

Reading a file (read sequence)



formatSelectorFilterFile

This selector is discussed in more detail on the previous page. The rest of this sequence will be called only if your plug–in module returns noErr from this call, and your module is selected by the plug–in host to process the file.

formatSelectorReadPrepare

This selector allows your plug–in module to adjust Photoshop's memory allocation algorithm. Photoshop sets **maxData** to the maximum number of bytes it can allocate to your plug–in. You may want to reduce **maxData** for increased efficiency. Refer to chapter 3 for details on memory management strategies.

formatSelectorReadStart

This selector allows the plug–in module to begin its interaction with the host.

You should initialize **imageMode**, **imageSize**, **depth**, **planes**, **imageHRes** and **imageVRes**. If an indexed color image is being opened, you should also set **redLUT**, **greenLUT** and **blueLUT**. If your plug–in has a block of image

resources you wish to have processed, you should read it in from the file and set **imageRsrcData** to be a handle to the resource data. See chapter 10 for more information about Photoshop image resources.

Your plug–in should allocate and read the first pixel image data buffer as appropriate. The area of the image being returned to the plug–in host is specified by **theRect**, **loPlane**, and **hiPlane**. The actual pixel data is pointed by **data**. The **colBytes**, **rowBytes**, **planeBytes**, and **planeMap** fields must specify the organization of the data.

Photoshop is very flexible in the format in which image data can be read. Here are two examples.

(1) To return just the red plane of an RGB color image, **loPlane** and **hiPlane** should be set to 0, **colBytes** should be set to 1, and **rowBytes** should be set to the width of the area being returned (**planeBytes** is ignored in this case, since **loPlane** == **hiPlane**).

(2) To return the RGB data in interleaved form (RGBRGB...), **loPlane** should be set to 0, **hiPlane** to 2, **planeBytes** to 1, **colBytes** to 3, and **rowBytes** to 3 times the width of area being returned.

formatSelectorReadContinue

This selector may be used to process a sequence of areas within the image. Your handler should process any incoming data and then, just as with the start call, set up **theRect**, **loPlane**, **hiPlane**, **planeMap**, **data**, **colBytes**, **rowBytes**, and **planeBytes** to describe the next chunk of the image being returned. The host will keep calling with formatSelectorReadContinue until you set **data** to NULL.

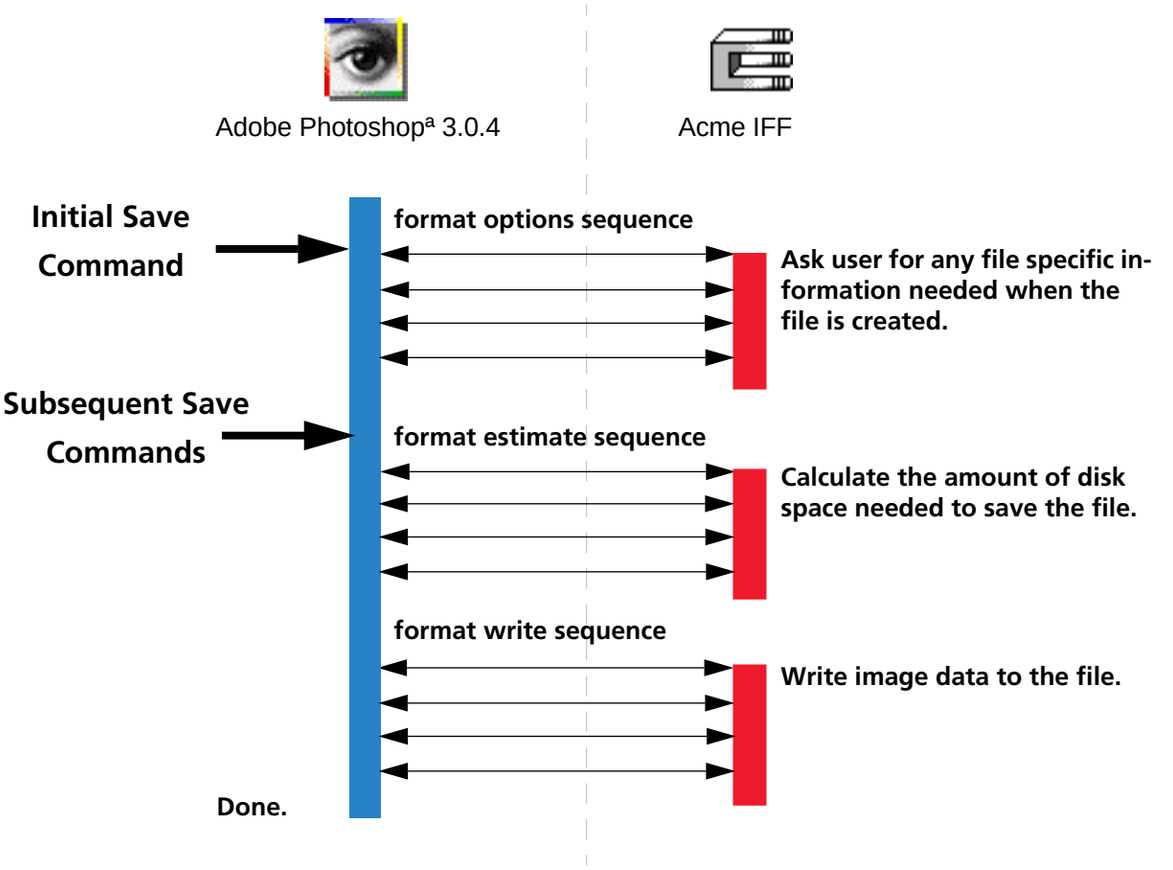
formatSelectorReadFinish

The formatSelectorReadFInish selector allows you to clean–up from the read operation just performed. This call is made by the plug–in host if and only if formatSelectorReadStart returned without error, even if one of the format-SelectorReadContinue calls results in an error.

Most plug–ins will at least need to free the buffer used to return pixel data if this has not been done previously.

If Photoshop detects Command–period (Mac OS) or Escape (Windows) while processing the results of a formatSelectorReadContinue call, it will call the formatSelectorReadFinish routine. Be careful here, since normally the plug–in would be expecting another formatSelectorReadContinue call.

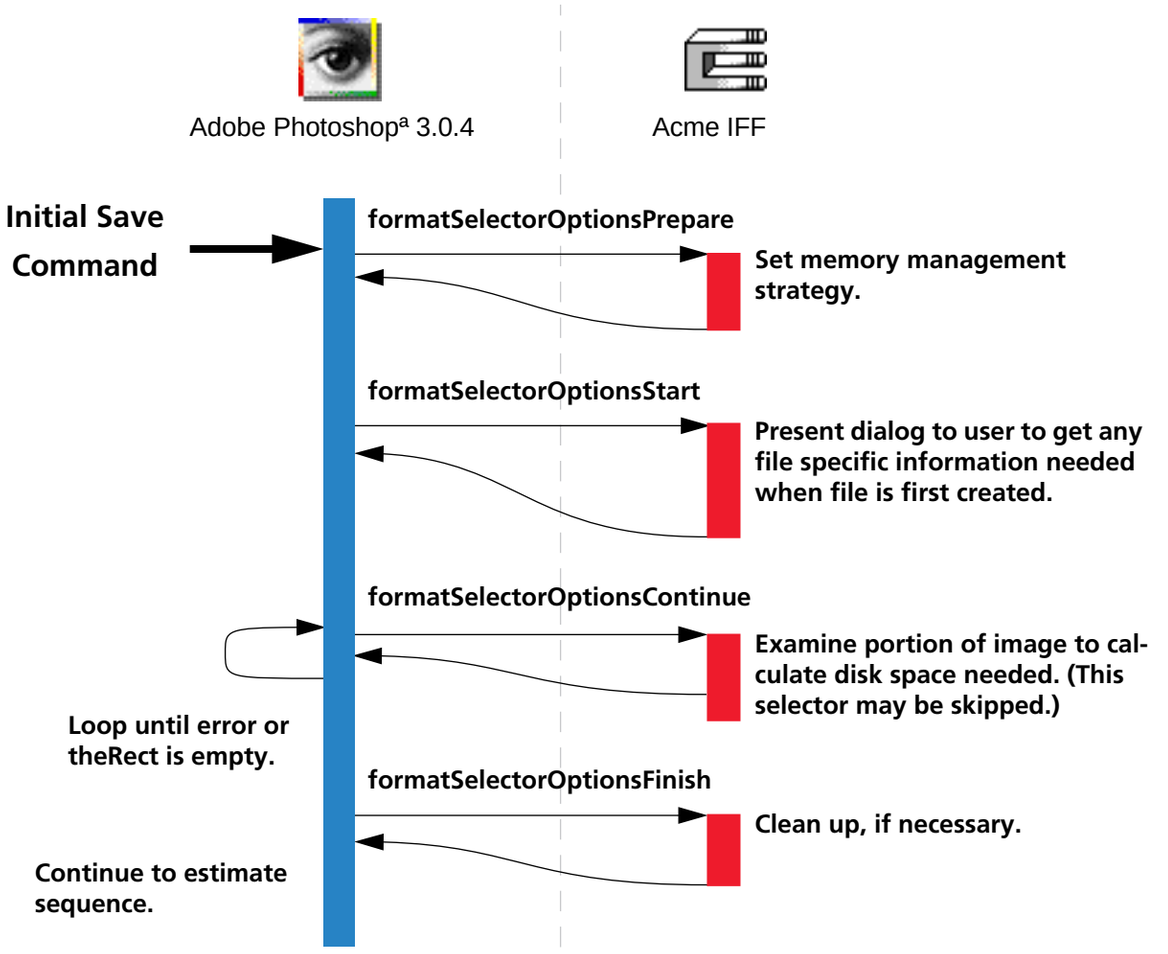
Writing a file



Writing a file involves either two or three distinct sequences, each similar in structure. When a document is first saved, Photoshop calls your format plug–in module with the **options sequence,** followed by the **estimate sequence** and the **write sequence**. After a document has been saved once, each time the user saves the file again, only the estimate and write sequences are called.

These sequences are discussed in more detail on the following pages.

Writing a file (options sequence)



formatSelectorOptionsPrepare

The formatSelectorOptionsPrepare selector call allows your plug–in module to adjust Photoshop's memory allocation algorithm. Photoshop sets **maxData** to the maximum number of bytes it can allocate to your plug–in. You may want to reduce **maxData** for increased efficiency. Refer to chapter 3 for details on memory management strategies.

formatSelectorOptionsStart

The formatSelectorOptionsStart selector call allows you to determine whether the current document can be saved in your file format, and if necessary, get any file options from the user.

If you need to examine the image to compute the file size, you can iterate through the image data (in formatSelectorOptionsContinue) in the same fashion as is done when writing the file to request sections of the image.

formatSelectorOptionsContinue

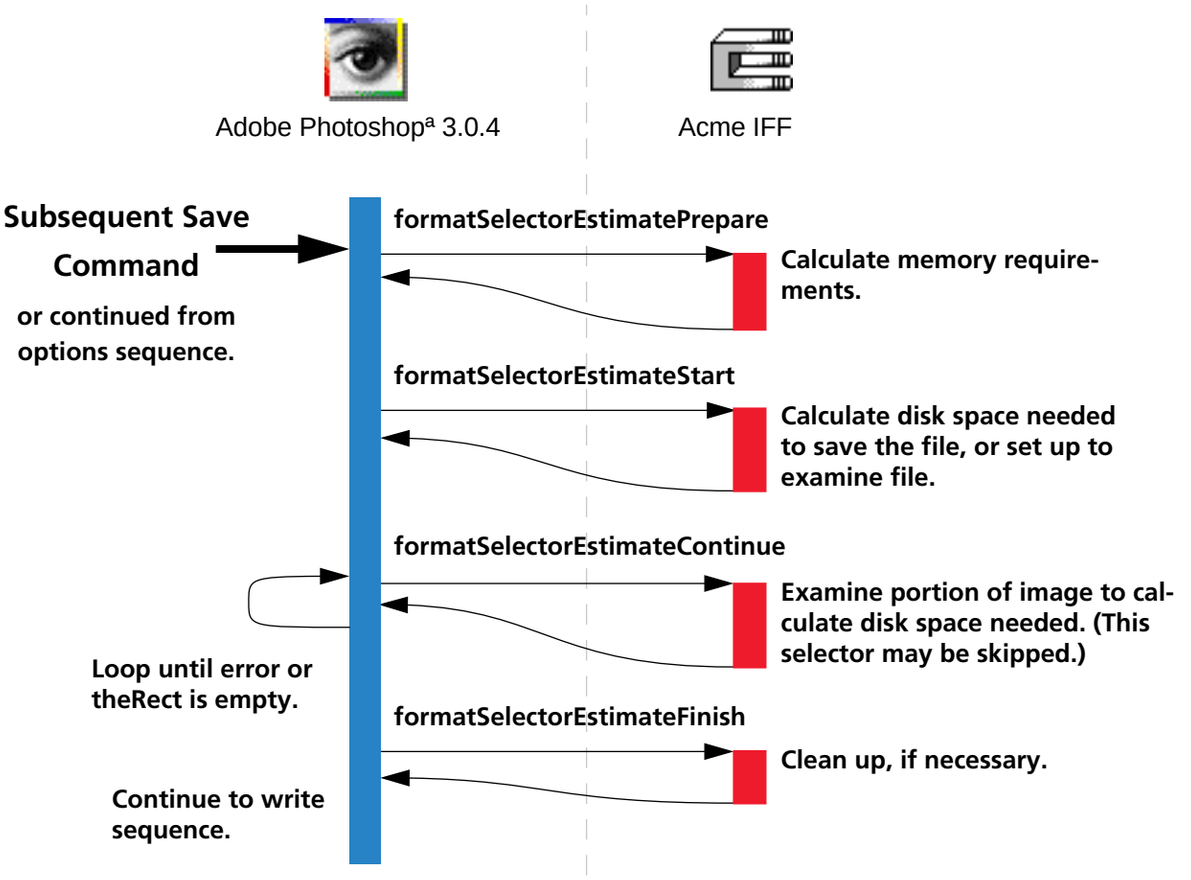
If the **data** field in the FormatRecord is set to NULL in the formatSelectorOptionsStart call, this selector will not be called at all. Otherwise, your plug–in can request parts of the image from which you determine whether your plug–in module can store the file. Refer to formatSelectorWriteStart and formatSelectorWriteContinue on the following page for details.

You may also use the AdvanceStateProc to iterate through the image.

formatSelectorOptionsFinish

Perform any clean up, if necessary.

Writing a file (estimate sequence)



formatSelectorEstimatePrepare

The formatSelectorWritePrepare selector call allows your plug–in module to adjust Photoshop's memory allocation algorithm. Photoshop sets **maxData** to the maximum number of bytes it can allocate to your plug–in. You may want to reduce **maxData** for increased efficiency. Refer to chapter 3 for details on memory management strategies.

formatSelectorEstimateStart

Calculate the disk space needed to save the file. If you can calculate the file size without examining the image data, you can set the **minDataBytes** and **maxDataBytes** fields in the FormatRecord to the approximate size of your file (due to compression, you may not be able to exactly calculate the final size), and set **data** to NULL.

If you need to examine the image to compute the file size, you can iterate through the image data (in formatSelectorEstimateContinue) in the same fashion as is done when writing the file to request sections of the image.

formatSelectorEstimateContinue

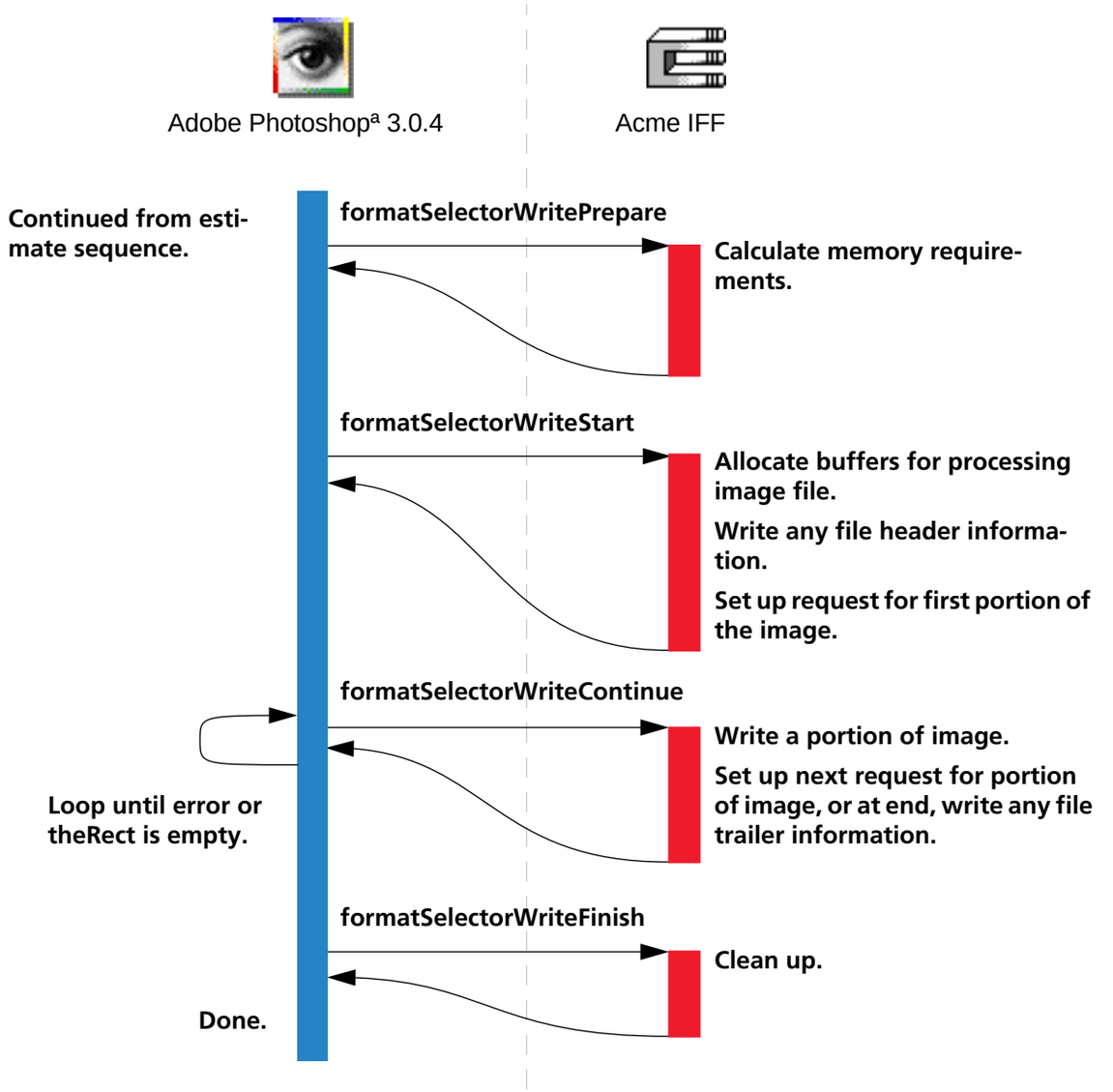
If the **data** field in the FormatRecord is set to NULL in the formatSelectorEstimateStart call, this selector will not be called at all. Otherwise, your plug–in can request parts of the image from which you can compute the minimum and maximum bytes to store the file. Refer to formatSelectorWriteStart and formatSelectorWriteContinue on the following page for details.

You may also use the AdvanceStateProc to iterate through the image.

formatSelectorEstimateFinish

Perform any clean up, if necessary.

Writing a file (write sequence)



formatSelectorWritePrepare

The formatSelectorWritePrepare selector call allows your plug–in module to adjust Photoshop's memory allocation algorithm. Photoshop sets **maxData** to the maximum number of bytes it can allocate to your plug–in. You may want to reduce **maxData** for increased efficiency. Refer to chapter 3 for details on memory management strategies.

formatSelectorWriteStart

The formatSelectorReadStart selector call allows your plug–in module to begin writing the file. On entry, the file to be written is open, and the file pointer is positioned at the start of the file. You should write any file header information, such as image resources, to the file.

Your plug–in should then indicate which portion of the image data to provide for the first formatSelectorWriteContinue call. The area of the image being requested from the plug–in host is specified by **theRect**, **loPlane**, and **hiPlane**. The actual pixel data is pointed by **data**.

You must specify the organization of the data to be returned by the plug–in host in the **colBytes**, **rowBytes**, **planeBytes**, and **planeMap** fields. Photoshop is very flexible in the format in which image data can be delivered to the plug–in module. Here are two examples:

(1) To return just the red plane of an RGB color image, **loPlane** and **hiPlane** should be set to 0, **colBytes** should be set to 1, and **rowBytes** should be set to the width of the area being returned (**planeBytes** is ignored in this case, since **loPlane** == **hiPlane**).

(2) To return the RGB data in interleaved form (RGBRGB...), **loPlane** should be set to 0, **hiPlane** to 2, **planeBytes** to 1, **colBytes** to 3, and **rowBytes** to 3 times the width of area being returned.

formatSelectorWriteContinue

This selector is call repeatedly by the plug–in host to provide your plug–in module some or all of the image data; your plug–in module should write this data to file. If successful, set up **theRect**, **loPlane**, **hiPlane**, **planeMap**, **data**, **colBytes**, **rowBytes**, and **planeBytes** to describe the next chunk of the image being requested.

The host will keep calling your formatSelectorReadContinue handler until you set **theRect** to an empty rectangle. Before returning after the last image data has been written, write any file trailer information to the file.

formatSelectorWriteFinish

The formatSelectorWriteFinish selector allows you to clean–up from the write operation just performed. This call is made by the plug–in host if and only if formatSelectorWriteStart returned without error, even if one of the formatSelectorWriteContinue calls results in an error.

Most plug–ins will at least need to free the buffer used to hold pixel data if this has not been done previously.

If Photoshop detects Command–period (Mac OS) or Escape (Windows) while processing the results of a formatSelectorWriteContinue call, it will call the formatSelectorWriteFinish routine. Be careful here, since normally the plug–in would be expecting another formatSelectorWriteContinue call.

Image Resources

Photoshop documents can have other properties associated with them besides pixel data. For example, documents typically contain page setup information and pen tool paths.

Photoshop supports the concept of a block of data known as the image resources for a file. Format plug–in modules can store and retrieve this information if the file format definition allows for a place to put such an arbitrary block of data (e.g., a TIFF tag or a PicComment).

Error return values

The plug–in module may return standard operating system error codes, or report its own errors, in which case it can return any positive integer.

```
#define formatBadParameters –30500  // a problem with the module interface
#define formatCannotRead    –30501  // a problem interpreting file data
```

Note: When writing a file, if your plug–in module sets ***result** to any non–zero value, then no subsequent selector calls will be made by Photoshop. For example, if in your formatSelectorOptionsStart handler, you determine that the file cannot be saved, then none of the remaining options, estimate, or write selectors will be called.

The Format parameter block

The pluginParamBlock parameter passed to your plug–in module's entry point contains a pointer to a FormatRecord structure with the following fields. This structure is declared in PIFormat.h.

Table 9–1: FormatRecord structure

Type	Field	Description
int32	serialNumber	This field contains Adobe Photoshop's serial number. Plug–in modules can use this value for copy protection, if desired.
TestAbortProc	abortProc	This field contains a pointer to the TestAbort callback. See chapter 3 for more details.
ProgressProc	progressProc	This field contains a pointer to the UpdateProgress callback. This procedure should only be called during the actual main operation of the plug–in, not during long operations during the preliminary user interface. See chapter 3.
int32	maxData	Photoshop initializes this field to the maximum of number of bytes it can free up. The plug–in may reduce this value during the prepare routines. The continue routines should process the image in pieces no larger than maxData less the size of any large tables or scratch areas it has allocated.
int32	minDataBytes	These fields give the limits on the data fork space needed to write the file. The plug–in should set these during the estimate sequence of selector calls.
int32	maxDataBytes	
int32	minRsrcBytes	These fields give the limits on the resource fork space needed to write the file. The plug–in should set these during the estimate sequence of selector calls.
int32	maxRsrcBytes	
int32	dataFork	The reference number for the data fork of the file to be read during the read sequence or written during the write sequence. During the options and estimate sequences, this field is undefined. On Windows, this is the file handle of the file returned by OpenFile ( ) API.
int32	rsrcFork	The reference number for the resource fork of the file to be read during the read sequence or written during the write sequence. During the options and estimate sequences, this field is undefined. On Windows, this field is undefined.
int16	imageMode	The formatSelectorReadStart routine should set this field to inform Photoshop what mode image is being acquired (grayscale, RGB Color, etc.). See the header file for possible values. Photoshop will set this field before it calls formatSelectorOptionsStart, formatSelectorEstimateStart, or formatSelectorWriteStart.
Point	imageSize	The formatSelectorReadStart routine should set this field to inform Photoshop of the image's width (imageSize.h) and height (imageSize.v) in pixels. Photoshop will set this field before it calls formatSelectorOptionsStart, formatSelectorEstimateStart, or formatSelectorWriteStart.

Table 9–1: FormatRecord structure (Continued)

Type	Field	Description
int16	depth	The formatSelectorReadStart routine should set this field to inform Photoshop of the image's resolution in bits per pixel per plane. The only valid settings are 1 for bitmap mode images, and 8 for all other modes. Photoshop will set this field before it calls formatSelectorOptionsStart, formatSelectorEstimateStart, or formatSelectorWriteStart.
int16	planes	The formatSelectorReadStart routine should set this field to inform Photoshop of the number of channels in the image. For example, if an RGB image without alpha channels is being returned, this field should be set to 3. Photoshop will set this field before it calls formatSelectorOptionsStart, formatSelectorEstimateStart, or formatSelectorWriteStart. Because of the implementation of the plane map, format modules (and acquire modules) should never try to work with more than 16 planes at a time. The results would be unpredictable.
Fixed	imageHRes	The formatSelectorReadStart routine should set these fields to inform Photoshop of the image's horizontal and vertical resolution in terms of pixels per inch. This is a fixed point number (16 binary digits). Photoshop initializes these fields to 72 pixels per inch. Photoshop will set these fields before it calls formatSelectorOptionsStart, formatSelectorEstimateStart, or formatSelectorWriteStart. The current version of Photoshop only supports square pixels, so it ignores the imageVRes field. Plug–ins should set both fields anyway in case future versions of Photoshop support non–square pixels.
Fixed	imageVRes	
LookUpTable	redLUT	If an indexed color mode image is being returned, the formatSelectorReadStart routine should return the image's color table in these fields. If an indexed color document is being written, Photoshop will set these fields before it calls formatSelectorOptionsStart, formatSelectorEstimateStart, or formatSelectorWriteStart.
LookUpTable	greenLUT	
LookUpTable	blueLUT	
void *	data	The start and continue routines should return a pointer to the buffer where image data is or is to be stored in this field. After the entire image has been processed, the continue selectors should set this field to NULL. Note that the plug–in is responsible for freeing any memory pointed to by this field.
Rect	theRect	The plug–in should set this to the area of the image covered by the buffer specified in data.
int16	loPlane	The start and continue routines should set this to the first and last planes covered by the buffer specified in data. For example, if interleaved RGB data is being used, they should be set to 0 and 2, respectively.
int16	hiPlane	
int16	colBytes	The start and continue routines should set this field to the offset in bytes between columns of data in the buffer. This is usually 1 for non–interleaved data, or (hiPlane – loPlane + 1) for interleaved data.

Table 9–1: FormatRecord structure (Continued)

Type	Field	Description
int32	rowBytes	The start and continue routines should set this field to the offset in bytes between rows of data in the buffer.
int32	planeBytes	The start and continue routines should set this field to the offset in bytes between planes of data in the buffers. This field is ignored if loPlane = hiPlane. It should be set to 1 for interleaved data.
PlaneMap (array of 16 int16's)	planeMap	This is initialized by the host to a linear map (planeMap [i] = i). This is used to map plane (channel) numbers between the plug–in and the host. For example, Photoshop stores RGB images with an alpha channel in the order RGBA, whereas most frame buffers store the data in ARGB order. To work with the data in this order, the plug–in should set planeMap [0] to 3, planeMap [1] to 0, planeMap [2] to 1, and planeMap [3] to 2.
Boolean	canTranspose	If the host supports transposing images during or after reading or before or during writing, it should set this field to TRUE. Photoshop always sets this field to TRUE.
Boolean	needTranspose	Initialized by the host to FALSE. If the plug–in wishes to have the image transposed, and canTranspose is TRUE, it should set this field to TRUE during the start call.
OSType	hostSig	The plug–in host provides its signature to your plug–in module in this field. Photoshop's signature is '8BIM'.
HostProc	hostProc	If not NULL, this field contains a pointer to a host–defined callback procedure that can do anything the host wishes. Plug–ins should verify hostSig before calling this procedure. This provides a mechanism for hosts to extend the plug–in interface to support application specific features.
int16	hostModes	This field is used by the host to inform the plug–in which imageMode values it supports. If the corresponding bit (LSB = bit 0) is 1, the mode is supported. This field can be used by plug–ins to disable reading unsupported file formats.

Table 9–1: FormatRecord structure (Continued)

Type	Field	Description
Handle	revertInfo	This field is set to NULL by Photoshop when a format for a file is first created. If this field is defined on a formatSelectorReadStart call, then treat the call as a revert and don't query the user. If it is NULL on the formatSelectorReadStart call, then query the user as appropriate and set up this field to store a handle containing the information necessary to read the file without querying the user for additional parameters (essential for reverting the file) and if possible to write the file without querying the user. The contents of this field are sticky to a document and will be duplicated when we duplicate the image format information for a document. On all formatSelectorOptions calls, leave revertInfo containing enough information to revert the document. Photoshop will dispose of this field when it disposes of the document, hence, the plug–in must call on Photoshop to allocate the data as well using the following callbacks or the callbacks provided in the handle suite.
NewPIHandleProc	hostNewHdl	This is the same as the NewPIHandle callback described in chapter 3. This field existed before the handle suite was defined.
DisposePIHandleProc	hostDisposeHdl	This is the same as the DisposePIHandle callback described in chapter 3. This field existed before the handle suite was defined.
Handle	imageRsrcData	During calls to the write sequence, this field contains a handle to a block of data to be stored in the file as image resource data. Since this handle is allocated before the write sequence begins, plug–ins must add any resources they want saved to the document during the options or estimate sequence. Since options is not always called, the best time is during the estimate sequence. During the read sequence, Photoshop checks this field after each call to formatSelectorRead and formatSelectorContinue and the first time it is non–NULL parses the handle as a block of image resource data for the current document.
int32	imageRsrcSize	This is the size of the handle in imageRsrcData. It is really only relevant during the estimate sequence when it is provided instead of the actual resource data.
PlugInMonitor	monitor	This field contains the monitor setup information for the host. See Appendix A for more details.
void *	platformData	This field contains a pointer to platform specific data. Not used on the Macintosh.
BufferProcs *	bufferProcs	This field contains a pointer to the buffer suite if it is supported by the host, otherwise NULL.
ResourceProcs *	resourceProcs	This field contains a pointer to the pseudo–resource suite if it is supported by the host, otherwise NULL.
ProcessEventProc	processEvent	This field contains a pointer to the ProcessEvent callback documented in chapter 3. It contains NULL if the callback is not supported.

Table 9–1: FormatRecord structure (Continued)

Type	Field	Description
DisplayPixelsProc	displayPixels	This field contains a pointer to the DisplayPixels callback documented in chapter 3. It contains NULL if the callback is not supported.
HandleProcs	handleProcs	This field contains a pointer to the handle suite if it is supported by the host, otherwise NULL.
These fields are new in version 3.0 of Adobe Photoshop.		
OSType	fileType	This field contains the file type for filtering.
ColorServicesProc	colorServices	This field contains a pointer to the ColorServices callback documented in chapter 3. It contains NULL if the callback is not supported.
AdvanceStateProc	advanceState	The advanceState callback allows you to drive the interaction through the inner (formatSelectorOptionsContinue) loop without actually returning from the plug–in. If it returns an error, then the plug–in generally should treat this as an error formatSelectorOptionsContinue and pass it on when it returns. For more information, see chapter 3.
These fields are new in version 3.0.4 of Adobe Photoshop.		
PropertyProcs *	propertyProcs	A pointer to the property callback suite. See chapter 3 for details.
int16	tileWidth	The width of the tiles. Zero if not set.
int16	tileHeight	The height of the tiles. Zero if not set.
int16	tileOrigin	The origin point for the tiles.
char[220]	reserved	Set to zero by the host for future expansion of the plug–in standard. Must not be used by plug–ins.

10 Document File Formats

Adobe Photoshop saves a user's document in one of several formats, which are listed under the pop–up menu in the "Save" dialog. This chapter documents these standard formats.

The formats discussed in this chapter include Photoshop 3.0 native format, Photoshop EPS format, Filmstrip format, and TIFF format.

For more information about file formats, you may wish to consult the *Encyclopedia of Graphics File Formats* by James D. Murray & William vanRyper (1994, O'Reilly & Associates, Inc., Sebastopol, CA ISBN 1–56592–058–9).

Image resource blocks

Image resource blocks are the basic building unit of several file formats, including Photoshop's native file format, JPEG, and TIFF. Image resources are used to store non–pixel data associated with an image, such as pen tool paths. (They are referred to as resource data because they hold data that was stored in the Macintosh's resource fork in early versions of Photoshop.)

The basic structure of Image Resource Blocks is shown in table 10–1.

Table 10–1: Image resource block

Type	Name	Description
OSType	Type	Photoshop always uses its signature, '8BIM'
int16	ID	Unique identifier (see table 10–2).
PString	Name	A pascal string, padded to make size even (a null name consists of two bytes of 0)
int32	Size	Actual size of resource data. This does not include the Type, ID, Name, or Size fields.
Variable	Data	Resource data, padded to make size even

Image resources use several standard ID numbers, as shown in table 10–2. Not all file formats use all ID's. Some information may be stored in other sections of the file.

Table 10–2: Image resource ID's

ID	Description
0x03E8 (1000)	Obsolete—Photoshop 2.0 only. Contains five int16 values: number of channels, rows, columns, depth, and mode.
0x03E9 (1001)	Optional. Macintosh print manager print info record.
0x03EB (1003)	Obsolete—Photoshop 2.0 only. Contains the indexed color table.
0x03ED (1005)	Resolution information. See appendix A for description of the ResolutionInfo structure.
0x03EE (1006)	Names of the alpha channels as a series of Pascal strings.
0x03EF (1007)	Display information for each channel. See appendix A for a description of the DisplayInfo structure.
0x03F0 (1008)	Optional. The caption as a Pascal string.
0x03F1 (1009)	Border information. Contains a Fixed number for the border width, and an int16 for border units (1=inches, 2=cm, 3=points, 4=picas, 5=columns).
0x03F2 (1010)	Background color. See the Colors file information in chapter 11.
0x03F3 (1011)	Print flags. A series of one byte boolean values (see Page Setup dialog): labels, crop marks, color bars, registration marks, negative, flip, interpolate, caption.
0x03F4 (1012)	Grayscale and multichannel halftoning information.
0x03F5 (1013)	Color halftoning information.
0x03F6 (1014)	Duotone halftoning information.
0x03F7 (1015)	Grayscale and multichannel transfer function.

Table 10–2: Image resource ID's

ID	Description
0x03F8 (1016)	Color transfer functions.
0x03F9 (1017)	Duotone transfer functions.
0x03FA (1018)	Duotone image information.
0x03FB (1019)	Two bytes for the effective black and white values for the dot range.
0x03FC (1020)	Obsolete.
0x03FD (1021)	EPS options.
0x03FE (1022)	Quick Mask information. 2 bytes containing Quick Mask channel ID, 1 byte boolean indicating whether the mask was initially empty.
0x03FF (1023)	Obsolete.
0x0400 (1024)	Layer state information. 2 bytes containing the index of target layer. 0=bottom layer.
0x0401 (1025)	Working path (not saved). See path resource format later in this chapter.
0x0402 (1026)	Layers group information. 2 bytes per layer containing a group ID for the dragging groups. Layers in a group have the same group ID.
0x0403 (1027)	Obsolete.
0x0404 (1028)	IPTC-NAA record. This contains the File Info... information.
0x0405 (1029)	Image mode for raw format files.
0x0406 (1030)	JPEG quality. Private.
0x07D0–0x0BB6 (2000–2998)	Path Information (saved paths)
0x0BB7 (2999)	Name of clipping path
0x2710 (10000)	Print flags information. 2 bytes version (= 1), 1 byte center crop marks, 1 byte (always zero), 4 bytes bleed width value, 2 bytes bleed width scale.

Path resource format

Photoshop stores the paths saved with an image in an image resource block. These resource blocks consist of a series of 26 byte path point records, and so the resource length should always be a multiple of 26.

Photoshop stores its paths as resources of type '8BIM' with IDs in the range 2000 through 2999. These numbers should be reserved for Photoshop. The name of the resource is the name given to the path when it was saved.

If the file contains a resource of type '8BIM' with an ID of 2999, then this resource contains a Pascal–style string containing the name of the clipping path to use with this image when saving it as an EPS file.

The path format returned by GetProperty ( ) call is identical to what is described below. Refer to the "Paths To Illustrator" sample plug–in code to see how this resource data is constructed.

Path points

All points used in defining a path are stored in eight bytes as a pair of 32–bit components, vertical component first.

The two components are signed, fixed point numbers with 8 bits before the binary point and 24 bits after the binary point. Three guard bits are reserved in the points to eliminate most concerns over arithmetic overflow. Hence, the range for each component is 0xF0000000 to 0x0FFFFFFF representing a range of –16 to 16. The lower bound is included, but not the upper bound.

This limited range is used because the points are expressed relative to the image size. The vertical component is given with respect to the image height, and the horizontal component is given with respect to the image width. <0,0> represents the top–left corner of the image; <1,1> (<0x01000000,0x01000000>) represents the bottom–right.

On Intel processors (Windows), the byte order of the path point components are reversed; you should swap the bytes when accessing each 32–bit value.

Path records

The data in a path resource consists of one or more 26 byte records. The first two bytes of each record is a selector to indicate what kind of path data record it is. Under Windows, you should swap the bytes before accessing it as a short (int16).

Table 10–3: Path data record types

Selector	Description
0	Closed subpath length record
1	Closed subpath Bezier knot, linked
2	Closed subpath Bezier knot, unlinked
3	Open subpath length record
4	Open subpath Bezier knot, linked
5	Open subpath Bezier knot, unlinked
6	Path fill rule record
7	Clipboard record

The first 26 byte path record contains a selector value of 6 (path fill rule record), the remaining 24 bytes of the first record are all zeroes which indicates that paths use even/odd rule. Subpath length records (selector value 0 or 3) contain the number of Bezier knot records in bytes 2 and 3. The remaining 22 bytes are unused, and should be zeroes. Each length record is then immediately followed by the Bezier knot records describing the knots of the subpath.

In Bezier knot records, the 24 bytes following the selector field contain three path points (described above) for:

(1) the control point for the Bezier segment preceding the knot,

(2) the anchor point for the knot, and

(3) the control point for the Bezier segment leaving the knot.

Linked knots have their control points linked; editing one point edits the other one to preserve collinearity. Knots should only be marked as having linked controls if their control points are collinear with their anchor. The control points on unlinked knots are independent of each other. Refer to the Adobe Photoshop User Guide for more information.

Clipboard records, selector = 7, contain four Fixed point numbers for the bounding rectangle (top, left, bottom, right), and a single Fixed point number indicating the resolution.

Photoshop 3.0 files

Macintosh file type: '8BPS'
Windows file extension:.PSD

This is the native file format for Adobe Photoshop 3.0. It supports storing all layer information.

Photoshop 3.0 files under Windows

All data is stored in big endian byte order; under Windows you must byte swap short and long integers when reading or writing.

Photoshop 3.0 files under Mac OS

For cross–platform compatibility, all information needed by Adobe Photoshop 3.0 is stored in the data fork. For interoperability with other Macintosh applications, however, some information is duplicated in resources stored in the resource fork of the file.

For compatibility with Adobe Fetch, a 'pnot' 0 resource contains references to thumbnail, keywords, and caption information stored in other resources. The thumbnail picture is stored in a 'PICT' resource, the keywords are stored in a 'STR#' 128 resource and the caption text is stored in a 'TEXT' 128 resource. For more information on the format of these resources see *Inside Macintosh: QuickTime Components* and the *Adobe Fetch Awareness Developer's Toolkit.*

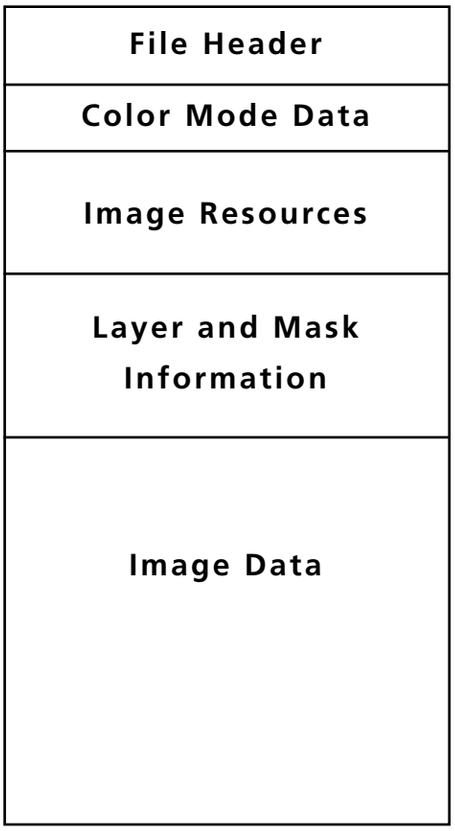
All of the data from Photoshop's File Info dialog is stored in an 'ANPA' 10000 resource. The data in this resource is stored as an IPTC–NAA record 2 and should be readable by various tools from Iron Mike. For more information on the format of this resource contact:

IPTC–NAA Digital Newsphoto Parameter Record
Newspaper Association of America
The Newspaper Center
11600 Sunrise Valley Drive
Reston VA 20091

Photoshop also creates 'icl8' –16455 and 'ICN#' –16455 resources containing thumbnail images which will be shown in the Finder.

Photoshop 3.0 file format

The file format for Photoshop 3.0 is divided into five major parts.



The file header is fixed length, the other four sections are variable in length.

When writing one of these sections, you should write all fields in the section, as Photoshop may try to read the entire section. Whenever writing a file and skipping bytes, you should explicitly write zeros for the skipped fields.

When reading one of the length delimited sections, use the length field to decide when you should stop reading. In most cases, the length field indicates the number of bytes, not records following.

File header section

The file header contains the basic properties of the image.

Table 10–4: File header

Length	Name	Description
4 bytes	Signature	Always equal to '8BPS'. Do not try to read the file if the signature does not match this value.
2 bytes	Version	Always equal to 1. Do not try to read the file if the version does not match this value.
6 bytes	Reserved	Must be zero.
2 bytes	Channels	The number of channels in the image, including any alpha channels. Supported range is 1 to 24.
4 bytes	Rows	The height of the image in pixels. Supported range is 1 to 30,000.

Table 10–4: File header (Continued)

Length	Name	Description
4 bytes	Columns	The width of the image in pixels. Supported range is 1 to 30,000.
2 bytes	Depth	The number of bits per channel. Supported values are 1, 8, and 16.
2 bytes	Mode	The color mode of the file. Supported values are: Bitmap = 0 Grayscale = 1 Indexed Color = 2 RGB Color = 3 CMYK Color = 4 Multichannel = 7 Duotone = 8 Lab Color = 9

Color mode data section

Only indexed color and duotone have color mode data. For all other modes, this section is just 4 bytes: the length field, which is set to zero.

For indexed color images, the length will be equal to 768, and the color data will contain the color table for the image, in non–interleaved order.

For duotone images, the color data will contain the duotone specification, the format of which is not documented. Other applications that read Photoshop files can treat a duotone image as a grayscale image, and just preserve the contents of the duotone information when reading and writing the file.

Table 10–5: Color mode data

Length	Name	Description
4 bytes	Length	The length of the following color data.
Variable	Color data	The color data.

Image resources section

The third section of the file contains image resources. As with the color mode data, the section is indicated by a length field followed by the data. The image resources in this data area are described in detail earlier in this chapter.

Table 10–6: Image resources

Length	Name	Description
4 bytes	Length	Length of image resource section.
Variable	Resources	Image resources.

Layer and mask information section

The fourth section contains information about Photoshop 3.0 layers and masks. The formats of these records are discussed later in this chapter. If

there are no layers or masks, this section is just 4 bytes: the length field, which is set to zero.

Table 10–7: Layer and mask information

Length	Name	Description
4 bytes	Length	Length of the miscellaneous information section.
Variable	Layers	Layer info. See table 10–10.
Variable	Masks	One or more layer mask info structures. See table 10–13.

Image data section

The image pixel data is the last section of a Photoshop 3.0 file. Image data is stored in planar order, first all the red data, then all the green data, etc. Each plane is stored in scanline order, with no pad bytes.

If the compression code is 0, the image data is just the raw image data.

If the compression code is 1, the image data starts with the byte counts for all the scan lines (rows * channels), with each count stored as a two–byte value. The RLE compressed data follows, with each scan line compressed separately. The RLE compression is the same compression algorithm used by the Macintosh ROM routine PackBits, and the TIFF standard.

Table 10–8: Image data

Length	Name	Description
2 bytes	Compression	Compression method. Raw data = 0, RLE compressed = 1.
Variable	Data	The image data.

Layer and mask records

Information about each layer and mask in a document is stored in the fourth section of the file. The complete, merged image data is not stored here; it resides in the last section of the file.

The first part of this section of the file contains layer information, which is divided into layer structures and layer pixel data, as shown in table 10–9. The second part of this section contains layer mask data, which is described in table 10–16.

Table 10–9: Layer info section

Length	Name	Description
4 bytes	Length	Length of the layers info section, rounded up to a multiple of 2.
Variable	Layers structure	Data about each layer in the document. See table 10–10.
Variable	Pixel data	Channel image data for each channel in the order listed in the layers structure section. See table 10–15.

Table 10–10: Layer structure

Length	Name	Description
2 bytes	Count	Number of layers. If <0, then number of layers is absolute value, and the first alpha channel contains the transparency data for the merged result.
Variable	Layer	Information about each layer (table 10–11).

Table 10–11: Layer records

Length	Name	Description
4 bytes	Layer top	The rectangle containing the contents of the layer.
4 bytes	Layer left	
4 bytes	Layer bottom	
4 bytes	Layer right	
2 bytes	Number channels	The number of channels in the layer.
Variable	Channel length info	Channel information. This contains a six byte record for each channel. See table 10–12.
4 bytes	Blend mode signature	Always ‘8BIM’.

Table 10–11: Layer records (Continued)

Length	Name	Description
4 bytes	Blend mode key	'norm' = normal 'dark' = darken 'lite' = lighten 'hue ' = hue 'sat ' = saturation 'colr' = color 'lum ' = luminosity 'mul ' = multiply 'scrn' = screen 'diss' = dissolve 'over' = overlay 'hLit' = hard light 'sLit' = soft light 'diff' = difference
1 byte	Opacity	0 = transparent ... 255 = opaque
1 byte	Clipping	0 = base, 1 = non–base
1 byte	Flags	bit0: transparency protected bit1: visible
1 byte	(filler)	(zero)
4 bytes	Extra data size	Length of the extra data field. This is the total length of the next five fields.
24 bytes, or 4 bytes if no layer mask.	Layer mask data	See table 10–13.
Variable	Layer blending ranges	See table 10–14.
Variable	Layer name	Pascal string, padded to a multiple of 4 bytes.

Table 10–12: Channel length info

Length	Name	Description
2 bytes	Channel ID	0 = red, 1 = green, etc. –1 = transparency mask –2 = user supplied layer mask
4 bytes	Length	Length of following channel data.

Table 10–13: Layer mask data

Length	Name	Description
4 bytes	Size	Size of layer mask data. This will be either 0x14, or zero (in which case the following fields are not present).
4 bytes	Top	Rectangle enclosing layer mask.
4 bytes	Left	
4 bytes	Bottom	
4 bytes	Right	

Table 10–13: Layer mask data

Length	Name	Description
1 byte	Default color	0 or 255
1 byte	Flags	bit 0: position relative to layer bit 1: layer mask disabled bit 2: invert layer mask when blending
2 bytes	Padding	Zeros

Table 10–14: Layer blending ranges data

Length	Name	Description
4 bytes	Length	Length of layer blending ranges data
4 bytes	Composite gray blend source	Contains 2 black values followed by 2 white values. Present but irrelevant for Lab & Grayscale.
4 bytes	Composite gray blend destination	Destination Range
4 bytes	First channel source range	
4 bytes	First channel destination range	
4 bytes	Second channel source range	
4 bytes	Second channel destination range	
...	...	...
4 bytes	Nth channel source range	
4 bytes	Nth channel destination range	

Table 10–15: Channel image data

Length	Name	Description
2 bytes	Compression	0 = Raw Data, 1 = RLE compressed.
Variable	Image data	If the compression code is 0, the image data is just the raw image data calculated as ((Layer Bottom – Layer Top) * (Layer Right – Layer Left)). If the compression code is 1, the image data starts with the byte counts for all the scan lines in the channel (Layer Bottom – Layer Top), with each count stored as a two–byte value. The RLE compressed data follows, with each scan line compressed separately. The RLE compression is the same compression algorithm used by the Macintosh ROM routine PackBits, and the TIFF standard. If the data since the Layers Size is odd, a pad byte will be inserted.

Table 10–16: Layer mask data

Length	Name	Description
2 bytes	Overlay color space	
8 bytes	Color components	4 * 2 byte color components
2 bytes	Opacity	0 = transparent, 100 = opaque.
1 byte	Kind	0 = Color selected—i.e. inverted 1 = Color protected 128 = use value stored per layer. This value is preferred. The others are for backward compatibility with beta versions.
1 byte	(filler)	(zero)

Photoshop EPS files

Photoshop 3.0 writes a high–resolution bounding box comment to the EPS file immediately following the traditional EPS bounding box comment. The comment begins with "%%HiResBoundingBox" and is followed by four numbers identical to those given for the bounding box except that they can have fractional components (i.e., a decimal point and digits after it). The traditional bounding box is written as the rounded version of the high resolution bounding box for compatibility.

Photoshop writes its image resources out to a block of data stored as follows:

```
%BeginPhotoshop: <length> <hex data>
```

<length> is the length of the image resource data.

<hex data> is the image resource data in hexadecimal.

Photoshop includes a comment in the EPS files it writes so that it is able to read them back in again. Third party programs that write pixel–based EPS files may want to include this comment in their EPS files, so Photoshop can read their files.

The comment must follow immediately after the %% comment block at the start of the file. The comment is:

```
%ImageData: <columns> <rows> <depth> <mode> <pad channels> <block size>
<binary/hex> "<data start>"
```

<columns> is the width of the image in pixels.

<rows> is the height of the image in pixels.

<depth> is the number of bits per channel. Must be 1 or 8.

<mode> is the image mode. 1 for bitmap and gray scale images (determined by depth), 2 for Lab images, 3 for RGB images, and 4 for CMYK images.

<pad channels> is the number of other channels stored in the file, which are ignored when reading. (Photoshop uses this to include a gray scale image that is printed on non–color PostScript printers).

<block size> is the number of bytes per row per channel. This will be equal to (<columns> * <depth> + 7) / 8 if the data is stored in line–interleave format (or if there is only one channel), or equal to 1 of the data is interleaved.

<binary/hex> is 1 if the data is in binary format, and 2 if the data is in hex format.

<data start> contains the entire PostScript line immediately preceding the image data. (This entire line should not occur elsewhere in the PostScript header code, but it may occur at part of a line.)

Filmstrip files

Adobe Premiere 2.0 supports the filmstrip file format. Premiere users can export any video clip as a filmstrip. Refer to the *Adobe Premiere User Guide* for more information.

Adobe Photoshop 3.0 supports the filmstrip file type to allow each frame to be individually painted. The filmstrip file format is fairly simple, and is described in this section.

A filmstrip consists of a sequence of equal sized 32–bit images, known as frames. The channel order in the file is Red, Green, Blue, Alpha.

After each frame is an arbitrarily sized leader area, in which any type of information may be embedded. Adobe Premiere puts the timecode and frame number for the frame in this area. This area is ignored by Premiere when the file is read.

Following all the frames is a 16 row trailer frame (it has the same width as the other frames). Adobe Premiere writes a yellow and black diagonal pattern in this area. The lower right corner of this area is actually an information record that exists at the very end of the file. This record is located by seeking to the end of the file minus the size of the record, then reading the record and verifying the signature field that it contains.

```
// Definition for filmstrip info record

typedef struct {
    long                signature;   // 'Rand'
    long                numFrames;   // number of frames in file
    short               packing;     // packing method
    short               reserved;    // reserved, should be 0
    short               width;       // image width
    short               height;      // image height
    short               leading;     // horiz gap between frames
    short               framesPerSec;// frame rate
    char                spare[16];   // some spare data.
} FilmStripRec, **FilmStripHand;
```

Table 10–17: FilmStripRec structure

Type	Field	Description
long	signature	This field must be set to the code 'Rand' and is used to verify the validity of the record.
long	numFrames	This is the total number of frames in the file.
short	packing	This is the packing method used, currently only a value of 0 is defined, for no packing.
short	width	The width of each image, in pixels.
short	height	The height of each image, in pixels.
short	leading	The height of the leading areas, in pixels.
short	framesPerSec	The rate at which the frames should be played.

To locate the filmstrip info record, seek to the end of the file minus (sizeof(FilmStripRec)), then read in the FilmStrip record. Check the signature field for the code 'Rand' to test for validity.

To locate the data for a particular frame, seek to (frame * width * (height+leading) * 4), then read (width * height * 4) bytes. If the data is being placed into a GWorld (Mac OS), the channels must be re–arranged from Red–Green–Blue–Alpha to Alpha–Red–Green–Blue.

To write a FilmStrip file, write each frame sequentially into the file, including the leading areas. Then write a block of ((width * (height+leading) * 4) – sizeof(FilmStripRec)) bytes. Finally, fill in and write the FilmStrip record to the file.

Note: The packing field should currently be zero. In the future packing methods may be defined for filmstrips, so any software which reads film-strips should examine this field before opening the file.

TIFF files

The same image resources information found in Photoshop 3.0 files are stored in TIFF files under tag number 34377 (see Image Resource Blocks and Image Resources earlier in this chapter).

For TIFF files the caption data is stored in an image description tag 270 and all the information is stored as an IPTC–NAA record 2 in tag 33723. The tag number was chosen by inspecting files written by Iron Mike software, and is supposed to be defined in a Rich TIFF specification. The tag is also specified in:

NSK TIFF
The Japan Newspaper Publishers & Editors Association
Nippon Press Center Building
2–2–1 Uchlsaiwai–cho
Chiyoda–ku, Tokyo 100

For more information about the TIFF format see:

TIFF Revision 6.0
(206) 628–5693

In reading the files, the following order is used with information read lower on the list replacing information read higher.

Image Description Tag (TIFF only)
IPTC–NAA Tag (TIFF only)

It is a bug that the TIFF information comes prior to the image resource information on this list. This means that an edit to the TIFF info will not be recognized unless the image resource information is removed. The TIFF data may be moved to after the image resource information in a future version of Photoshop.

Table 10–16 describes the standard TIFF tags and tag values that Photoshop 3.0 is able to read and write.

TIFF files under Mac OS

For cross–platform compatibility, all TIFF information is stored in the data fork. For interoperability with other Macintosh applications, however, some information is duplicated in resources stored in the resource fork of the file.

For compatibility with Adobe Fetch, a 'pnot' 0 resource contains references to thumbnail, keywords, and caption information stored in other resources. The thumbnail picture is stored in a 'PICT' resource, the keywords are stored in a 'STR#' 128 resource and the caption text is stored in a 'TEXT' 128 resource. For more information on the format of these resources see *Inside Macintosh: QuickTime Components* and the *Adobe Fetch Awareness Developer's Toolkit.*

All of the data from Photoshop's File Info dialog is stored in an 'ANPA' 10000 resource. The TIFF file also contains a 'STR ' -16396 resource that contains the string "Adobe Photoshop™ 3.0" which indicates the application that created the TIFF file.

Photoshop also creates 'icl8' –16455 and 'ICN#' –16455 resources containing thumbnail images which will be shown in the Finder.

Table 10–18: TIFF Tags

Tag	Photoshop reads	Photoshop writes
IFD	First IFD in file	Only one IFD per file
NewSubFileType	Ignored	0
ImageWidth	1 to 30000	1 to 30000
ImageLength	1 to 30000	1 to 30000
BitsPerSample	1, 2, 4, 8, 16 (all same)	1, 8, 16
Compression	1, 2, 5, 32773	1, 5
PhotometricInterpretation	0, 1, 2, 3, 5, 8	0 (1–bit), 1 (8–bit), 2, 3,5,8
FillOrder	1	No
ImageDescription	Printing Caption	Printing Caption
StripOffsets	Yes	Yes
SamplesPerPixel	1 to 24	1 to 24
RowsPerStrip	Any	Single strip if not compressed, multiple strips if compressed.
StripByteCounts	Required if compressed	Yes
XResolution	Yes	Yes
YResolution	Ignored (square pixels assumed)	Yes
PlanarConfiguration	1 or 2	1
ResolutionUnit	2 or 3	2
Predictor	1 or 2	1 or 2
ColorMap	Yes	Yes
TileWidth	Yes	No
TileLength	Yes	No
TileOffsets	Yes	No
TileByteCounts	Required if compressed	No
InkSet	1	No
DotRange	Yes, if CMYK	Yes
ExtraSamples	Ignored (except for count)	0

11 Load File Formats

Besides documents that the user creates in Adobe Photoshop (discussed in chapter 10), there are a number of other files used by Photoshop to store information about colors, brushes, etc. These can be saved to files and loaded into Photoshop at a later time, even for use in a different image. These are referred to generically as “load files”.

Each load file has a unique file type and file extension associated with it. Photoshop for Macintosh will recognize either, but does not require the use of the extension. Photoshop for Windows will look for the given file extension automatically; this can be overridden.

Many, but not all, of the files have version numbers written as short integers in the first two bytes of the file.

On the Macintosh, all information is stored in the data forks of Photoshop’s load files. The files are completely interchangable with Photoshop for Windows or any other platform.

Note that this requires consistent byte ordering between the all platforms when reading and writing these files. Photoshop stores multi–byte values with the high–order bytes first (big–endian, used on 680x0 systems), the reverse of the way this is done on Intel platforms (little–endian).

Arbitrary Map

Macintosh file type: '8BLT'
Windows file extension:.AMP

Arbitrary Map files are loaded and saved in Photoshop's Curves dialog.

There is no version number written in the file. The file must be an even multiple of 256 bytes long.

Each 256 bytes is a lookup table, where the first byte corresponds to zero in the image data and the last byte to 255 in the image data. A "null" table that has no effect on an image is a linear table of bytes from 0 to 255.

If there is one table in the file, Photoshop applies it to the master composite channel, if the image has one, or to the single active channel if there is only one. If there is no composite channel, but more than one active channel, the load operation will have no effect. If the file has exactly three tables then it is assumed to represent an RGB lookup table and they are applied to the first channels in the image (the master composite map is untouched). If there is a single active channel, then the RGB lookup table is converted to gray-scale and the result is applied to the active channel. In any other case, the first map is treated as a master and the remainder are applied to the image channels in turn (i.e. the second map is associated with the first channel, the third map with the second channel, etc.)

Photoshop handles single active channels in a special fashion. When saving the map applied to a single channel, only one map is written to the file. Similarly, when reading a map file for application to a single active channel, the master map is the one that will be used on that channel. This allows easy application of a single file to both composite and Grayscale images.

Brushes

Macintosh file type: '8BBR'
Windows file extension:.ABR

Brushes settings files are loaded and saved in Photoshop's Brushes palette. These are typically stored in the "Goodies:Brushes & Patterns" sub–folder (Mac OS), or BRUSHES sub–directory (Windows).

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Count (2 bytes)
A short integer indicating how many brushes are in the remainder of the file.

3. Brushes (variable)
Two types of brushes are currently supported: elliptical, computed brushes and sampled brushes. Computed brushes are created with the New Brush command; sampled brushes are created from selected image data using the Define Brush command.

Each brush contains the following components:

a. type (2 bytes)
A short integer indicating the type of brush. A value of 1 means a computed brush, a value of 2 means a sampled brush. Other values are currently undefined.

b. size (4 bytes)
A long integer indicating the number of bytes in the remainder of the brush definition. Photoshop uses this information to skip over brush types that it doesn't understand.

c. data (size bytes)
The contents depend on the type of brush. Computed brush data is always 14 bytes; sampled brush data varies in size depending on the image data that makes up the brush tip.

Computed brushes:

i. miscellaneous (4 bytes): a long value which is ignored.

ii. spacing (2 bytes): a short integer ranging from 0 to 999 (0 means no spacing)

iii. diameter (2 bytes): a short integer ranging from 1 to 999

iv. roundness (2 bytes): a short integer ranging from 0 to 100

v. angle (2 bytes): a short integer ranging from –180 to 180

vi. hardness (2 bytes): a short integer ranging from 0 to 100

Sampled brushes:

i. miscellaneous (4 bytes): a long value which is ignored.

ii. spacing (2 bytes): a short integer ranging from 0 to 999 (0 means no spacing)

iii. anti–aliasing (1 byte): indicates whether the brush is to be anti–aliased when applied; 0 means no anti–aliasing. (Note that brushes with

sampled data size either taller or wider than 32 pixels will not be anti–aliased by Photoshop in any event.)

iv. bounds (8 bytes): a rectangle, four short integers giving the bounds of the sampled tip data (in the order top, left, bottom, right)

v. bounds2 (16 bytes): a rectangle, exactly repeating the previous bounds entry, but in four long integers instead of four short integers.

vi. depth (2 bytes): depth of the sampled data, which is always 8

vii. image data (variable): if the bounds are taller than 16384, the data is broken into 16384–line chunks. Each chunk is streamed as follows:

   a. compression (2 bytes): two values are currently defined: 0 = Raw Data, 1 = RLE compressed

   b. data (variable): the brush tip image data is a single plane of grayscale data, stored in scanline order, with no pad bytes.

If the compression code is 0, the data is just the raw image data.

If the compression code is 1, the data starts with the byte counts for all the scan lines (equal to the number of rows, as described by the bounds), with each count stored as a two–byte value. The RLE compressed data follows, with each scan line compressed separately. The RLE compression is the same compression algorithm used by the Macintosh ROM routine PackBits, and the TIFF standard.

Color Table

Macintosh file type: '8BCT'
Windows file extension:.ACT

Color Table files are loaded and saved in Photoshop's Color Table dialog (used with Indexed Color images), and can be loaded into the Colors palette as well.

There is no version number written in the file. The file is exactly 768 bytes long.

This file contains 256 RGB colors, starting with the first color in the table (index 0), with three bytes per color, in the order red, green, blue.

If loaded into the Colors palette, the colors will be installed in the color swatch list as RGB colors.

Colors

Macintosh file type: '8BCO'
Windows file extension:.ACO

Colors files are loaded and saved in Photoshop's Colors palette. These are typically stored in the PALETTES sub–directory (Windows)

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Count (2 bytes)
A short integer indicating how many colors are in the remainder of the file.

3. Colors (Count * 10 bytes)
Each color is ten bytes in size, and is made up of the following subsections:

a. color space ID (2 bytes)
A short integer indicating the color space the color belongs to as shown in table 1.

b. color data (8 bytes)
Four short integers (possibly unsigned) that are the actual color data. If the color does not require four values to specify, the extra values are undefined and should be written as zeroes. The most basic color spaces are outlined below.

Table 11–1: Color data

Color ID	Name	Description
0	RGB	The first three values in the color data are, respectively, the color's red, green, and blue components. They are full unsigned 16–bit values as in Apple's RGBColor data structure (e.g. pure red is defined as 65535, 0, 0).
1	HSB	The first three values in the color data are, respectively, the color's hue, saturation, and brightness components. They are full unsigned 16–bit values as in Apple's HSV-Color data structure (e.g. pure red is defined as 0, 65535, 65535).
2	CMYK	The four values in the color data are, respectively, the color's cyan, magenta, yellow, and black components. They are full unsigned 16–bit values, with 0 representing 100% ink (e.g. pure cyan is defined as 0, 65535, 65535, 65535).
7	Lab	The first three values in the color data are, respectively, the color's lightness, a chrominance, and b chrominance components. The lightness component is a 16–bit value ranging from 0 to 10000. The chromanance components are each 16–bit values ranging from –12800 to 12700. Gray values are represented by chrominance components of 0 (e.g. pure white is defined as 10000, 0, 0).
8	Grayscale	The first value in the color data is the gray value; it ranges from 0 to 10000.

Photoshop allows the specification of custom colors, such as those colors that are defined in a set of custom inks provided by a printing ink manufacturer. These colors can be stored in the Colors palette and streamed to and from load files. The details of a custom color's color data fields are not public and should be treated as a black box.

Table 2 gives the color space IDs currently defined by Photoshop for some custom color spaces:.

Table 11–2: Custom color spaces

Color ID	Name
4	FOCOLTONE COLOUR SYSTEM
10	HKS colors (European Photoshop only)
3	PANTONE MATCHING SYSTEM
6	TOYO 88 COLORFINDER 1050
5	TRUMATCH color

Command Settings File

Macintosh file type: '8BFK'
Windows file extension:.ACM

Commands settings files are loaded and saved in Photoshop 3.0's Commands palette. This feature supplants the Function Key feature of Photoshop 2.5. The Commands palette buttons are simple mappings to Photoshop menu items, with optional function key shortcut and colorization.

1. Version (2 bytes)
Equal to 2, written as a short integer.

2. Count (2 bytes)
The number of command records that follow. There are no pad bytes between records.

3. Command Records (variable)
The remainder of the file contains the Command records, one after the other. Each one is composed of the following:

a. Command ID (4 bytes)
This field is obsolete and must be set to zero.

b. Function Key ID (2 bytes)
This is an integer ranging from –15 to 15. Positive numbers map directly onto the numbered function keys (F1, F2, etc.) that are present on many personal computer keyboards. Negative numbers indicate that the shift key must be used as well for the keyboard shortcut (Shift–F1, Shift–F2, etc.). Zero means the button has no keyboard shortcut. On Windows systems, values outside of –12 to 12 will be ignored as standard Windows systems have 12 function keys on the keyboard. Windows systems will also map 1 to 0, as the F1 key is reserved for calling up Help. These numbers should be unique across all entries in a Commands file, however Photoshop will ignore duplicates.

c. Color Index (2 bytes)
Each command button can be assigned a color with which its background will be tinted when drawn. There are eight predefined colors, with matching values as follows: 0 = None (button drawn in black and white), 1 = Red, 2 = Orange, 3 = Yellow, 4 = Green, 5 = Blue, 6 = Purple, 7 = Gray.

d. Title Matching Flag (1 byte)
If set to 1, this boolean flag indicates that the button title should automatically be updated to match the command's current menu item text. For example, a button assigned to the Layers palette would change text from "Show Layers" to "Hide Layers" automatically as the state of the palette and the actual menu item changes. If set to 0, the button title has been changed from the menu item text by the user and shouldn't change unless changed by the user again.

e. Button Title (variable)
This is the title of the button that will be drawn on the Command palette. It usually matches the corresponding menu item text. It is stored as a Pascal–style string, with no pad bytes.

f. Command Key (variable)
This is the key for finding the menu item in Photoshop's menus. To distinguish menu items from each other, which could be duplicated on different menus, a key may include the title of the menu itself followed by a colon (e.g. "Mode:RGB Color"). This text is displayed in the options dialog for the button, but not on the Commands palette itself. (Note that even if the Title

Matching flag is turned on, the title of the button text on the screen never contains the menu title qualifier.) It is stored as a Pascal–style string, with no pad bytes.

Curves

Macintosh file type: '8BSC'
Windows file extension:.CRV

Curves settings files are loaded and saved in Photoshop's Curves dialog and Black Generation curve dialog (from within Separation Setup Preferences). Curves files can also be loaded into any of Photoshop's transfer function dialogs, such as the Duotone Curve dialog from within Duotone Options. (When loaded into a transfer function dialog, only the first curve in a Curves file is used.)

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Count (2 bytes)
A short integer indicating how many curves are in the file. Must be in the range 1 to 27.

3. Curves (variable)
The remainder of the file contains the curves, one after the other.

Each curve is written as follows (i.e. each curve is made up of the following subsections):

a. point count (2 bytes)
A short integer in the range 2 to 19 indicates how many points are in the curve.

b. curve points (point count * 2 bytes)
Each curve point is a pair of short integers where the first number is the output value (vertical coordinate on the Curves dialog graph) and the second is the input value. All coordinates have range 0 to 255.
Hence a null curve (no change to image data) is represented by the following five–number, ten–byte sequence in a file: 2 0 0 255 255 . (Note that Photoshop allows the option of displaying ink percentages instead of pixel values; this is a display option only and the internal data is unchanged, with 100% ink equal to image data of 0 and 0% ink equal to image data of 255.)

Generally, the first of the curves is a master curve that applies to all of the composite channels in a composite image mode (e.g. the Red, Green, and Blue channels are all modified by the master curve for an RGB document). The remaining curves apply to the active channels individually; the second curve applies to channel one (if it is an active channel), the third curve to channel two, etc., up until the seventeenth curve, which applies to channel sixteen. The exception to this, and the reason there are up to nineteen curves, is when the original image is indexed color. In this case three curves are created for the red, green, and blue portions of the image's color table, and they replace the curve that represents the first channel of the image. This adds two curves for indexed images, and so for indexed color images any alpha channel that is active corresponds to its channel number plus three (e.g. if channel two is active it corresponds to curve number 5).

Photoshop handles single active channels in a special fashion. When saving the curves applied to a single channel, the settings are stored into the master slot, at the beginning of the file. Similarly, when reading a curves file for application to a single active channel, the master curve is the one that will be used on that channel. This allows easy application of a single file to both RGB and Grayscale images.

Note that Photoshop 3.0 can write Curves files that Photoshop 2 will not be able to read, because Photoshop 3.0's active channel support is different from Photoshop 2.0's, and there could be more active channels in a Curves dialog than 2.0 supported. Photoshop 3.0 will always write at least five curves to a curves file, for maximum compatability with version 2.0. However, beyond the curve for the fourth channel, it does not write null curves past the last non–null curve that has been specified in the dialog. The presence of extraneous null curves will not affect a load operation.

Also note that it is possible to create a Curves load file with Photoshop 3.0 that cannot be read by Photoshop 2.5; Photoshop 3.0 allows a maximum of 24 channels per document, Photoshop 2.5 allows 16. Such use of the Curves function is rare, however.

Duotone options

Macintosh file type: '8BDT'
Windows file extension:.ADO

Duotone settings files are loaded and saved in Photoshop's Duotone Options dialog.

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Count (2 bytes)
A short integer indicating how many plates are in the duotone spec: 1 for monotones, 2 for duotones, 3 for tritones, 4 for quadtones. Must be in the range 1 to 4.

3. Ink Colors (4 * 10 bytes)
Four ink colors, regardless of the number of plates. The contents of the colors beyond the last plate specified by Count are undefined. Each color is streamed in the same fashion as in the Colors load file, and consists of the following subsections:

a. color space (2 bytes)
A short integer indicating the color space the color is in.

b. color data (8 bytes)
Four short integers (possibly unsigned) that are the actual color data.

Please refer to the Colors file format for details on the contents of the color records.

4. Ink Names (4 * 64 bytes)
Four ink names, regardless of the number of plates. Each name is streamed as a Pascal–style string with a length byte followed by the characters in the string. Names may not be more than 63 characters in length. Each name is padded to occupy 64 bytes including the initial length byte. Any names beyond the last plate specified by Count should be the empty string (size = 0).

5. Ink Curves (4 * 28 bytes)
Four ink curves, regardless of the number of plates. Each curve has the following subsections:

a. transfer curve (26 bytes)
An array of 13 short integers, each ranging from 0 to 1000 (representing 0.0 to 100.0). In addition, all but the first and last value may be –1 (representing no point on the curve). Hence a null transfer curve looks like this: 0, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, 1000.

b. override (2 bytes)
For compatability with Photoshop 2.0, this short integer should be 0. It is ignored by Photoshop 3.0.

Any curves beyond the last plate specified by Count should be equal to the null curve.

6. Dot Gain (2 bytes)
For compatability with Photoshop 2.0, this short integer should be 20. It is ignored by Photoshop 3.0.

7. Overprint Colors (11 * 10 bytes)
Eleven ink colors, regardless of the number of plates. The number of defined overprints depends on the number of plates, Count. For monotones, there are no overprint colors. For duotones, there is one overprint color. For tritones, there are four overprint colors. For quadtones, there are 11 overprint colors. The contents of the colors beyond the last defined overprint are undefined. Each color is streamed in the same fashion as in the Colors load file, and consists of the following subsections:

a. color space (2 bytes)
A short integer indicating the color space the color is in.

b. color data (8 bytes)
Four short integers (possibly unsigned) that are the actual color data.

Halftone screens

Macintosh file type: '8BHS'
Windows file extension:.AHS

Halftone Screens settings files are loaded and saved in Photoshop's Halftone Screens dialog (from within Page Setup).

1. Version (2 bytes)
Equal to 5, written as a short integer.

2. Screens (4 * 18 bytes)
Four screen descriptions, each of which has the following subsections:

a. frequency value (4 bytes)
This ink's screen frequency, in lines per inch. This is a binary fixed point value with sixteen bits representing each of the integer and fractional parts of the number. Values range from 1.0 to 999.999, with units in lpi (lines per inch).

b. frequency scale (2 bytes)
The units for the screen frequency. Line per inch = 1, lines per centimeter = 2. Does not affect the frequency value itself, merely the way the value will be displayed on the screen.

c. angle (4 bytes)
Angle for this screen, a binary fixed point value with sixteen bits representing each of the integer and fractional parts of the number. Values range from –180.0000 to 180.0000, measured in degrees.

d. shape code (2 bytes)
A code representing the shape of the halftone dots in this screen. Round = 0, Ellipse = 1, Line = 2, Square = 3, Cross = 4, Diamond = 6. Custom shapes are represented by a negative number. The absolute value of this number is the size in bytes of the custom Spot Function, which is outlined below.

e. miscellaneous (4 bytes)
For compatability, this should be set to 0. It is not currently used by Photoshop.

f. accurate screens (1 byte)
Boolean flag which is true (1) if accurate screens should be used, false (0) otherwise.

g. default screens (1 byte)
Boolean flag which is true (1) if printer's default screens should be used, false (0) otherwise.

3. Spot Functions (size is the sum of the absolute values of all negative shape codes)
For every screen which has a custom spot function, the text of the PostScript function is written here. The functions are written one after the other with no header information, in the same order as the screen settings (screen description 1's spot function, if it has one, followed by number 2's, etc.). The shape code for those screens that have custom functions provides enough information to separate the various functions and assign them.

Hue/Saturation

Macintosh file type: '8BHA'
Windows file extension:.HSS

Hue/Saturation settings files are loaded and saved in Photoshop's Hue/Saturation dialog.

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Mode (1 byte)
Photoshop's Hue/Saturation dialog has two overall modes: in one, the settings represent shifts in the image data's hue and saturation, in the other the entire image is colorized to a single hue. This byte is a boolean flag indicating whether the colorization data or the hue–adjustment data in the file should be used. If the byte is zero, the hue–adjustment data will be used. If the byte is non–zero (Photoshop writes it as a 1) the colorization data will be used. (Both sets of data are present, but only one is used depending on the value of this byte.)

3. Padding (1 byte)
This pad byte must be present but is ignored by Photoshop.

4. Colorization (6 bytes)
Three short integers representing colorization settings. All values are in the range –100 to 100. The first number is the hue in which the image data will be colorized; the user interface represents the range of values as –180 to 180, where the number represents the hue in the traditional HSB color wheel, with zero equal to red. The next number is the saturation, the third number is the lightness adjustment.

5. Hue–Saturation Settings (42 bytes)
This data consists of three sets of seven short integers; all values range from –100 to 100:

a. hue settings (14 bytes)
One master value and six other values. The first value is the master hue change. For RGB and CMYK images, the other six values apply to each of the six hextants in the HSB color wheel: those image pixels nearest to red, yellow, green, cyan, blue, or magenta. (These numbers appear in the user interface as being in the range –60 to 60; the values are nevertheless stored as –100 to 100 and the slider will reflect each of the possible 201 values.) For Lab images, the first four of these values are applied to image pixels in the four Lab color quadrants (yellow, green, blue, magenta), and the other two values are ignored (Photoshop sets them to zero). (The values that are used range from –90 to 90 in the user interface.)

b. saturation settings (14 bytes)
Seven short integers representing the saturation adjustments. The first is a master value. The other six are applied to pixels using the same hue sextant or quadrant breakdown as for the hue adjusments; as before the last two are ignored for Lab documents.

c. lightness settings (14 bytes)
The last seven short integers are the lightness adjustments. The first is a master value. The other six are applied to pixels using the same hue sextant or quadrant breakdown as for the hue and saturation adjusments; as before the last two are ignored for Lab documents.

Ink colors setup

Macintosh file type: '8BIC'
Windows file extension:.API

Ink Colors settings files are loaded and saved in Photoshop's Ink Colors Setup dialog, via the Preferences submenu.

1. Version (2 bytes)
Equal to 4, written as a short integer.

2. Ink Colors (27 * 2 bytes)
Nine short integer triples specifying the xyY (CIE) values for the inks and their combinations. The inks are specified in the order Cyan, Magenta, Yellow, Magenta–Yellow (Red), Cyan–Yellow (Green), Cyan–Magenta (Blue), Cyan–Magenta–Yellow, followed by the White and Black points. Each triple is written in the order x (range: 0 to 10000, representing 0.0 to 1.0000), y (range: 1 to 10000, representing 0.0001 to 1.0000), Y (range: o to 20000, representing 0.00 to 200.00).

3. Gray Balance (4 * 2 bytes)
Four short integers specifying the gray color balance for Cyan, Magenta, Yellow, and Black. Each ranges from 50 to 200 (representing 0.5 to 2.00).

4. Dot Gain (2 bytes)
A short integers specifying the dot gain. Ranges from –10 to 40 (–10% to 40%).

Custom kernel

Macintosh file type: '8BCK'
Windows file extension:.ACF

Kernel settings files are loaded and saved in Photoshop's Custom filter dialog.

There is no version number written in the file. The file is expected to be exactly 54 bytes long, representing 27 short integers.

1. Weights (50 bytes)
The first 25 values are the custom weights, applied to pixels offset from (–2, –2) to (2, 2) off of each image pixel. The values progress through horizontal offsets first, e.g. the first five values all represent a vertical offset of –2. Each value can range from –999 to 999.

2. Scale (2 bytes)
This value can range from 1 to 9999.

3. Offset (2 bytes)
This value can range from –9999 to 9999.

Levels

Macintosh file type: '8BLS'
Windows file extension:.ALV

Levels settings files are loaded and saved in Photoshop's Levels dialog. There are two versions of this file format. Photoshop 3.0 reads both but only writes version 2. Note that because the maximum number of channels that a document can contain was increased in Photoshop 3.0 (from 16 to 24), Photoshop 3.0 actually writes a longer Levels file than Photoshop 2.5. Photoshop 2.5 is still capable of reading these files, however, and will simply ignore the extra data.

1. Version (2 bytes)
Equal to 2, written as a short integer.

2. Levels Records (290 bytes)
This consists of 29 sets of levels. Each set of levels contains five short integers, in ten bytes.

The first number in a set is the input floor setting, and must range from 0 to 253. The second number is the input ceiling, and must range from 2 to 255. Third is the output value to which the input floor willbe matched. It can range from 0 to 255. Fourth is the ceiling output, also ranging from 0 to 255. The fifth value is the gamma to be applied to the image data. It ranges from 10 to 999 (representing the values 0.1 to 9.99).

The first set of levels are the master levels that apply to all of the composite channels in a composite image mode (e.g. the Red, Green, and Blue channels are all modified by the master levels settings for an RGB document).

The remaining sets apply to the active channels individually; the second set applies to channel one (if it is an active channel), the third set to channel two, etc., up until the 25th set, which applies to channel 24.

The exception to this is when the original image is indexed color. In this case three sets of levels are created for the red, green, and blue portions of the image's color table, and they replace the levels that represent the first channel of the image. This adds two sets of levels for indexed images, and so for indexed color images any alpha channel that is active corresponds to its channel number plus three (e.g. if channel two is active it corresponds to set number 5). The 28th and 29th sets are reserved for future use and should be set to zeroes.

Photoshop handles single active channels in a special fashion. When saving the levels applied to a single channel, the settings are stored into the master slot, at the beginning of the file. Similarly, when reading a levels file for application to a single active channel, the master levels are the ones that will be used on that channel. This allows easy application of a single file to both RGB and Grayscale images.

Monitor setup

Macintosh file type: '8BMS'
Windows file extension:.AMS

Monitor settings files are loaded and saved in Photoshop's Monitor Setup dialog, via the Preferences submenu.

1. Version (2 bytes)
Equal to 2, written as a short integer.

2. Gamma (2 bytes)
A short integer indicating the monitor's gamma. Must be in the range 75 to 300 (representing 0.75 to 3.00).

3. White Point (2 * 2 bytes)
Two short integers giving the monitor's white point: the first is the x value, ranging from 0 to 10000 (representing 0.0 to 1.0000), the second is the y value, ranging from 1 to 10000 (representing 0.0001 to 1.0000).

4. Phosphors (6 * 2 bytes)
Three sets of two integers giving the x–y coordinates of the red, green, and blue phosphors. First comes red x, then red y; then green x, etc. The x values range from 0 to 10000 (representing 0.0 to 1.0000); the y values range from 1 to 10000 (representing 0.0001 to 1.0000).

Replace color/Color range

Macintosh file type: '8BXT'
Windows file extension:.AXT

Replace Color settings files are loaded and saved in Photoshop's Replace Color dialog. They are also used to load and save settings from the Color Range dialog

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Color Space (2 bytes)
A short integer indicating what space the color components are in. 7 indicates Lab color, 8 indicates Grayscale. No other values are supported.

3. Component Ranges (6 bytes)
These six unsigned byte values represent the range of colors within which a pixel's color must fall to be considered selected for color replacement, or color range selecting. If the Color Space is grayscale, the first two bytes are the low and high endpoints of the range of gray values that are to be selected. The other four bytes should be zeroed. If the Color Space is Lab, then the first two bytes are the low and high endpoints of a range of 'L' values, the second two bytes are the low and high endpoints of a range of 'a' chromanance values, and the third pair bytes are the low and high endpoints of a range of 'b' chromanance values.

4. Fuzziness (2 bytes)
This short integer records the fuzziness setting, which controls how colors close to the selected colors are to be affected. It ranges from 0 to 200.

5. Transform Settings (6 bytes)
For files loaded into the Color Range dialog, these values are ignored. The Color Range dialog will write zeroes here. For Replace Color, this consists of three short integers; all values range from –100 to 100:

a. hue transform (2 bytes)
The hue change to be applied to the selected colors.

b. saturation transform (2 bytes)
The saturation change to be applied to the selected colors.

c. lightness transform (2 bytes)
The lightness change to be applied to the selected colors.

Scratch Area

Macintosh file type: '8BSR'
Windows file extension:.ASR

Scratch Area settings files are loaded and saved in Photoshop's Scratch palette.

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Scratch Area data (variable)
The Photoshop scratch area consists of RGB image data. The three planes of data are written one after the other, in the order Red, Green, Blue; each consists of the following:

a. bounds (16 bytes)
A rectangle, four long integers giving the bounds of the scratch data (in the order top, left, bottom, right); for Photoshop 3.0, this must always correspond to [0, 0, 89, 200] as the Scratch palette has a fixed size.

b. depth (2 bytes)
depth of the current plane of data, which is always 8.

c. image data (variable):

i. compression (2 bytes)
Two values are currently defined: 0 = Raw Data, 1 = RLE compressed

ii. data (variable)
Each plane of the scratch image data is stored in scanline order, with no pad bytes.

If the compression code is 0, the data is just the raw image data.

If the compression code is 1, the data starts with the byte counts for all the scan lines (equal to the number of rows, as described by the bounds), with each count stored as a two–byte value. The RLE compressed data follows, with each scan line compressed separately. The RLE compression is the same compression algorithm used by the Macintosh ROM routine PackBits, and the TIFF standard.

Selective color

Macintosh file type: '8BSV'
Windows file extension:.ASV

Selective Color settings files are loaded and saved in Photoshop's Selective Color dialog.

1. Version (2 bytes)
Equal to 1, written as a short integer.

2. Correction Method (2 bytes)
A short integer indicating how the color correction is to be applied: in Relative (0) or Absolute (1) mode.

3. Plate Corrections (80 bytes)
The remainder of the file contains 10 correction records, one after the other.

Each record is written as follows:

a. cyan correction (2 bytes)
A short integer in the range –100 to 100 indicating the amount of correction for the cyan component.

b. magenta correction (2 bytes)
A short integer in the range –100 to 100 indicating the amount of correction for the magenta component.

c. yellow correction (2 bytes)
A short integer in the range –100 to 100 indicating the amount of correction for the yellow component.

d. black correction (2 bytes)
A short integer in the range –100 to 100 indicating the amount of correction for the black component.

The first record is ignored by Photoshop 3.0 and is reserved for future use. It should be set to all zeroes. The rest of the records apply to specific areas of colors or lightness values in the image, in the following order: Reds, Yellows, Greens, Cyans, Blues, Magentas, Whites, Neutrals, Blacks.

Separation setup

Macintosh file type: '8BSS'
Windows file extension:.ASP

Separation settings files are loaded and saved in Photoshop's Separation Setup dialog, via the Preferences submenu.

1. Version (2 bytes)
Equal to 300, written as a short integer.

2. Separation Type (1 byte)
A boolean flag indicating UCR (value = 0) or GCR (value = 1) separations.

3. Black Limit (2 bytes)
A short integer giving the black ink limit, ranging from 0 to 100.

4. Total Limit (2 bytes)
A short integer giving the total ink limit, ranging from 200 to 400.

5. UCA Amount (2 bytes)
A short integer giving the undercolor addition for GCR separations, ranging from 0 to 100.

6. Black Generation Curve (variable)
This is a spline curve. The format is identical to a single curve instance from the Curves file format. It is composed of two parts:

a. point count (2 bytes)
A short integer in the range 2 to 19 indicates how many points are in the curve.

b. curve points (point count * 2 bytes)
Each curve point is a pair of short integers where the first number is the output value (vertical coordinate on the Black Generation dialog graph) and the second is the input value. All coordinates have range 0 to 255.

Hence a null curve (no change to input values) is represented by the following five–number, ten–byte sequence in a file: 2 0 0 255 255 .

Note that the black generation curve and the UCA limit must both be present even if the Separation Type is set to UCR.

Separation tables

Macintosh file type: '8BST'
Windows file extension:.AST

Separation Table files are loaded and saved in Photoshop's Separation Tables dialog.

If the size of the file is 33 * 33 * 33 * 4, then the file consists only of an Lab–>CMYK table as currently documented.

If the size of the file is 33 * 33 * 33 + 256 * 3, then the file consists only of a CMYK–>Lab table as currently documented.

Otherwise, the file has the following format:

1. Version (2 bytes)
Equal to 300, written as a short integer.

2. Has Lab to CMYK (1 byte)
Boolean indicating whether the file contains an Lab to CMYK table.

3. Has CMYK to Lab (1 byte)
Boolean indicating whether the file contains an CMYK to Lab table.

4. Lab to CMYK Table (33 * 33 * 33 * 4 bytes, optional)
If field 2 is equal to 1 (true), this section contains the CMYK colors for 33 * 33 *33 Lab colors. The Lab colors that are the source colors for this can be generated:

```
for (i = 0; i< 33; i++)
    for (j = 0; j < 33; j++)
        for (n = 0; n < 33; n++)
        {
            L = Min (i * 8, 255);
            a = Min (j * 8, 255);
            b = Min (n* 8, 255);
        }
```

The CMYK colors are written in interleaved order, one byte each ink, 0 = 100%, 255 = 0%.

5. CMYK to Lab Table ((33 * 33 * 33 + 256) * 3 bytes, optional)
If field 3 is equal to 1 (true), this section contains the Lab colors for 33 * 33 *33 + 256 CMYK colors. The CMYK colors that are the source colors for this can be generated:

```
for (i = 0; i< 33; i++)
    for (j = 0; j < 33; j++)
        for (n = 0; n < 33; n++)
        {
            c = Min (i * 8, 255);
            m = Min (j * 8, 255);
            y = Min (n* 8, 255);
            k = 255;
        }

for (i = 0; i < 256; i++)
{
    c = 255;
    m = 255;
    y = 255;
```

```
    k = i;
}
```

The Lab colors are written in interleaved order, one byte per component.

If after reading the above data, the file is not yet empty, then the file contains the following data.

6. Has Gamut Table (1 byte, 1 = have table, 0 = don't have table)
Flag to indicate whether a gamut table follows. If zero, then this flag is the last byte of the file; the filler byte and gamut table (described below) will not be present.

7. Filler (1 byte, optional)
If Has Gamut Table == 1, then this filler byte should also be set to 1 for compatibility reasons.

8. Gamut Table ((((33 * 33 * 33L) + 7) >> 3) bytes, optional)
If Has Gamut Table == 1, then the gamut table data consists of (((33 * 33 * 33L) + 7) >> 3) bytes of data. This is a bit table indexed in the same way as the Lab–>CMYK table with the provision that the high bit of the first byte is at index 0, etc.

To test the bit at bitIndex, use table [bitIndex >> 3] & (0x0080 >> (bitIndex & 0x07))) != 0. bitIndex itself is calculated in the same way you would calculate an index into the Lab–>CMYK table.

A 1 indicates that the color is in gamut and a 0 indicates that it is out of gamut.

Transfer function

Macintosh file type: '8BTF'
Windows file extension:.ATF

Transfer Function settings files are loaded and saved in Photoshop's Duotone Curve dialog (from within Duotone Options) and Transfer Function dialogs (from within Page Setup). Transfer Function files can also be loaded into any of Photoshop's curves dialogs, such as the Curves color adjustment dialog.

1. Version (2 bytes)
Equal to 4, written as a short integer.

2. Functions (112 bytes)
There are four transfer functions in the file. Each function is made up of the following subsections:

a. curve (26 bytes)
A transfer curve consists of 13 short integers, each ranging from 0 to 1000 (1000 represents the value 100.0). In addition, all but the first and last value may be –1 (representing no point on the curve). Hence a null transfer curve looks like this: 0, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, 1000.

b. override (2 bytes)
This is a boolean flag indicating whether the curve should override the printer's default transfer curve. If it is zero, the printer's curve will not be overridden.

Note again that the file always contains four functions. For example, when writing the printer transfer functions for Grayscale images Photoshop writes four copies of the single transfer function specified in the user interface.

A Data Structures

This appendix provides information about various data structures used by plug–in modules.

Information about the PiPL data structures in contained in chapter 4. PiMI data structures are described in chapter 5. The different plug–in parameter blocks are described in their respective chapters (6–9).

PSPixelMap

```
typedef struct PSPixelMap
{
    int32       version;
    VRect       bounds;
    int32       imageMode;
    int32       rowBytes;
    int32       colBytes;
    int32       planeBytes;
    void        *baseAddr;
    /* Fields new in version 1. */
    PSPixelMask  *mat;
    PSPixelMask  *masks;
    int32       maskPhaseRow;
    int32       maskPhaseCol;
} PSPixelMap;
```

Table A–1: PSPixelMap structure

Type	Field	Description
int32	version	The version number for this structure. The current version number is version 1. Future versions of Photoshop may support additional parameters and will support higher version numbers for PSPixelMap's.
VRect	bounds	The bounds for the pixel map.
int32	imageMode	The mode for the image data. The supported modes are grayscale, RGB, CMYK, and Lab. Additionally, if the mode of the document being processed is DuotoneMode or IndexedColorMode, you can pass plugInModeDuotone or plugInModeIndexedColor.
int32	rowBytes	The offset from one row to the next of pixels.
int32	colBytes	The offset from one column to the next of pixels.
int32	planeBytes	The offset from one plane of pixels to the next. In RGB, the planes are ordered red, green, blue; in CMYK, the planes are ordered cyan, magenta, yellow, black; in Lab, the planes are ordered L, a, b.
void *	baseAddr	The address of the byte value for the first plane of the top left pixel.
PSPixelMask *	mat	For all modes except indexed color, you can specify a mask to be used for matting correction. For example, if you have white matted data to display, you can specify a mask in this field which will be used to remove the white fringe. This field points to a PSPixelMask structure (see below) with a maskDescription indicating what type of matting needs to be compensated for. If this field is NULL, Photoshop performs no matting compensation. If the masks are chained, only the first mask in the chain is used.

Table A–1: PSPixelMap structure (Continued)

Type	Field	Description
PSPixelMask *	masks	This points to a chain of PSPixelMasks which are multiplied together (with the possibility of inversion) to establish which areas of the image are transparent and should have the checkerboard displayed. kSimplePSMask, kBlackMatPSMask, kWhiteMatPSMask, and kGrayMatPSMask all operate such that 255 = opaque and 0 = transparent. kInvertPSMask has 255 = transparent and 0 = opaque.
int32	maskPhaseRow	Tthe phase of the checkerboard with respect to the top left corner of the PSPixelMap.
int32	maskPhaseCol	The phase of the checkerboard with respect to the top left corner of the PSPixelMap.

PSPixelMask

The PSPixelMask structure is defined as follows:

```
typedef struct PSPixelMask
{
    struct PSPixelMask   * next
    void                 * maskData;
    int32                rowBytes;
    int32                colBytes;
    int32                maskDescription;

} PSPixelMask;
```

Table A–2: PSPixelMask structure

Type	Field	Description
`PSPixelMask *`	next	A pointer to the next mask in the chain
void *	maskData	A pointer to the mask data.
int32	rowBytes	The row step for the mask.
int32	colBytes	The column step for the mask.

int32	maskDescription	The mask description value, which is one of the following:
		
		`#define kSimplePSMask 0`
		`#define kBlackMatPSMask 1`
		`#define kGrayMatPSMask 2`
		`#define kWhiteMatPSMask 3`
		`#define kInvertPSMask 4`

ColorServicesInfo

This data structure is used in the ColorServices callback function. See chapter 3 and the notes following table A–3 for more details.

```
typedef struct ColorServicesInfo
    {
    int32 infoSize;
    int16 selector;
    int16 sourceSpace;
    int16 resultSpace;
    Boolean resultGamutInfoValid;
    Boolean resultInGamut;
    void *reservedSourceSpaceInfo;
    void *reservedResultSpaceInfo;
    int16 colorComponents[4];
    void *reserved;
    union
        {
        Str255 *pickerPrompt;
        Point *globalSamplePoint;
        int32 specialColorID;
        } selectorParameter;
    }
ColorServicesInfo;
```

Table A–3: ColorServicesInfo structure

Type	Field	Description
int32	infoSize	This field must be filled in with the size of the ColorServicesInfo record in bytes. The value is used as a version identifier in case this record is expanded in the future. It can be filled in like so: ColorServicesInfo requestInfo; requestInfo.infoSize = sizeof(requestInfo);
int16	selector	This field selects the operation performed by the ColorServices callback. #define plugIncolorServicesChooseColor 0 #define plugIncolorServicesConvertColor 1 #define plugIncolorServicesSamplePoint 2 #define plugIncolorServicesGetSpecialColor 3
int16	sourceSpace	This field is used for to indicate the color space of the input color contained in colorComponents. For plugIncolorServicesChooseColor the input color is used as an initial value for the picker. For plugIncolorServicesConvertColor the input color will be converted from the color space indicated by sourceSpace to the one indicated by resultSpace. Available color spaces: #define plugIncolorServicesRGBSpace 0 #define plugIncolorServicesHSBSpace 1 #define plugIncolorServicesCMYKSpace 2 #define plugIncolorServicesLabSpace 3 #define plugIncolorServicesGraySpace 4 #define plugIncolorServicesHSLSpace 5 #define plugIncolorServicesXYZSpace 6

Table A–3: ColorServicesInfo structure (Continued)

Type	Field	Description
int16	resultSpace	This field holds the desired color space of the result color from the ColorServices call. The result will be contained in the colorComponents field when ColorServices returns. For the plugIncolorServicesChooseColor selector, resultSpace can be set to plugIncolorServicesChosenSpace to return the color in whichever color space the user chose the color. In that case, resultSpace will contain the chosen color space on output.
Boolean	resultGamutInfoValid	This output only field indicates whether the resultInGamut field has been set. In Photoshop 3.0, this will only be true for colors returned in the plugIncolorServicesCMYKSpace color space.
Boolean	resultInGamut	This output only field is a boolean value that indicates whether the returned color is in gamut for the currently selected printing setup. It is only meaningful if the resultGamutInfoValid field is true.
void *	reservedSourceSpaceInfo	Must be NULL. A parameter error will be returned if they are not.
void *	reservedResultSpaceInfo	Must be NULL. A parameter error will be returned if they are not.
int16	colorComponents[4]	This array contains the actual color components of the input or output color. Refer to the notes following this table for more information about the color values that are represented in this array.
void *	reserved	Must be NULL. A parameter error will be returned if they are not.
union	selectorParameter	This union is used for providing different information based on the selector field. The pickerPrompt variant contains a pointer to a Pascal string which will be used as a prompt in the Photoshop color picker for the plugIncolorServicesChooseColor call. NULL can be passed to indicate no prompt should be used. The globalSamplePoint field points to a Point record that indicates the current sample point. The specialColorID should be either: plugIncolorServicesForegroundColor (0) or plugIncolorServicesBackgroundColor (1).

Notes:
The colorComponents array contains color values as listed in the color space name. Components not used in the input color space need not be filled in and components not used in the result color space are undefined.

For **RGB colors**, the colorComponents values are:

colorComponent[0] = Red value
colorComponent[1] = Green value
colorComponent[2] = Blue value
colorComponent[3] = (undefined)

Each RGB value should be in the range 0 through 255.

For **HSB colors**, the colorComponents values are:

colorComponent[0] = Hue value
colorComponent[1] = Saturation value
colorComponent[2] = Brightness value
colorComponent[3] = (undefined)

The Hue value should be in the range 0 through 359 degrees. The Saturation and Brightness values should be in the range 0 through 255 representing 0 through 100%.

For **L*a*b colors**, the colorComponents values are:

colorComponent[0] = L value
colorComponent[1] = a value
colorComponent[2] = b value
colorComponent[3] = (undefined)

L ranges from 0 to 255 representing the range 0...100.

a and b range from 0 to 255 representing the range -128 to 127. Note that these are merely shifted by +128, it is not a two's complement signed number.

For **CMYK colors**, the colorComponents values are:

colorComponent[0] = Cyan value
colorComponent[1] = Magenta value
colorComponent[2] = Yellow value
colorComponent[3] = blacK value

Each component ranges from 0 to 255, representing ink values from 100% down to 0%. This may be counterintuitive, since larger values represent lighter colors. This does make it easier to use CMY and RGB interchangeably.

For **Grayscale colors**, the colorComponents values are:

colorComponent[0] = Gray value
colorComponent[1] = (undefined)
colorComponent[2] = (undefined)
colorComponent[3] = (undefined)

The gray value should be in the range 0 through 255.

PlugInMonitor

A number of the plug–in module types get passed monitor descriptions via the PlugInMonitor structure. These descriptions basically detail the information recorded in Photoshop's Monitor Setup dialog and are passed in a structure of the following type:

```
typedef struct PlugInMonitor
{
      Fixed gamma;
      Fixed redX;
      Fixed redY;
      Fixed greenX;
      Fixed greenY;
      Fixed blueX;
      Fixed blueY;
      Fixed whiteX;
      Fixed whiteY;
      Fixed ambient;

} PlugInMonitor;
```

The fields of this record are as follow:

Table A–4: PlugInMonitor structure

Type	Field	Description
Fixed	gamma	This field contains the monitor's gamma value or zero if the whole record is invalid.
Fixed	redX	These fields specify the chromaticity coordinates of the monitor's phosphors.
Fixed	redY	
Fixed	greenX	
Fixed	greenY	
Fixed	blueX	
Fixed	blueY	
Fixed	whiteX	These fields specify the chromaticity coordinates of the monitor's white point.
Fixed	whiteY	
Fixed	ambient	This field specifies the relative amount of ambient light in the room. Zero means a relatively dark room, 0.5 means an average room, and 1.0 means a bright room.

ResolutionInfo

This structure contains information about the resolution of an image. It is written as an image resource. See chapter 10 for more details.

```
struct ResolutionInfo
    {
    Fixed               hRes;
    int16               hResUnit;
    int16               widthUnit;
    Fixed               vRes;
    int16               vResUnit;
    int16               heightUnit;
    };
```

hRes and **vRes** are always stored in units of pixels/inch. **hResUnit** and **vResUnit** are 1 if the resolution should be displayed in pixels/inch, or 2 if it should be displayed in units of pixels/cm.

widthUnit and **heightUnit** are 1 for inches, 2 for cm, 3 for points, 4 for picas, or 5 for columns.

DisplayInfo

This structure contains display information about each channel. It is written as an image resource. See chapter 10 for more details.

```
struct DisplayInfo
    {
    int16               colorSpace;
    int16               color[4];
    int16               opacity;   // 0..100
    char                kind;      // selected = 0, protected = 1
    char                padding;   // should be zero
    };
```

See the table 11–1 for a list of colorSpace values.

B PiPL Grammar

This information is included as reference material. If you use the example source code and the documentation earlier in the chapter, you probably won't need to worry about the specifics of the PiPL syntax.

```
# Miscellaneous definitions

<OSType>

<int16>

<int32>

<epsilon> :=

# Beginning of real grammar.

<PiPL spec> := <resource header> <resource body>

<resource header> :=
    "resource" "'PiPL'" "("
     <resourceID> <optional resource name> <optional attribute list>
    ")"

<optional name> :=
    <epsilon> |
    "," <string>

<optional attribute list> :=
    <epsilon> |
    "," <attribute> <attribute list tail>

<attribute list tail> :=
    <epsilon> |
     "|" <attribute> <attribute list tail>

<resource body> :=
    "{" "{"
    <property list>
    "}" "}"

<property list tail> :=
    <epsilon> |
    "," <property> <property list tail>

<property list> :=
    <epsilon>
    | <property> <property list tail>

<property> :=
    <kind property> |
    <version property> |
    <priority property> |
    <required host property> |
    <name property> |
```

```
    <category property> |
    <68k code descriptor property> |
    <powerpc code descriptor property> |
    <win32 x86 code property> |
    <supported modes property> |
    <filter case info property> |
    <format file type property> |
    <read types property> |
    <write types property> |
    <filtered types property> |
    <read extensions property> |
    <write extensions property> |
    <filtered extensions property> |
    <format flags property> |
    <format maximum size property> |
    <format maximum channels property> |
    <parsable types property> |
    <parsable extensions property> |
    <filtered parsable types property> |
    <filtered parsable extensions property> |
    <parsable clipboard types property>

<kind property> := "Kind" "{" <kind ID> "}"

<kind ID> := <OSType> |
    "Filter" |
    "Parser" |
    "ImageFormat" |
    "Extension" |
    "Acquire" |
    "Export"

<version property> := "Version" "{" <version clause> "}"

<version clause> := <int32> |
    "(" <wired version ID high> "<<" "16" ")" "|"
    "(" <wired version ID low> ")" |
    <wired version ID>

<wired version ID> := "FilterVersion" |
    "ParserVersion" |
    "ImageFormatVersion" |
    "ExtensionVersion" |
    "AcquireVersion" |
    "ExportVersion"

<wired version ID high> := "latestFilterVersion" |
    "latestParserVersion" |
    "latestImageFormatVersion" |
    "latestExtensionVersion" |
    "latestAcquireVersion" |
    "latestExportVersion"

<wired version ID high> := "latestFilterSubVersion" |
    "latestParserSubVersion" |
    "latestImageFormatSubVersion" |
    "latestExtensionSubVersion" |
    "latestAcquireSubVersion" |
    "latestExportSubVersion"

<priority property> := "Priority" "{" <int16> "}"

<required host property> := "Host" "{" <OSType> "}"
```

```
<name property> := "Name" "{" <string> "}"

<category property> := "Category" "{" <string> "}"

<68k code descriptor property> := "Code68k" "{" <OSType>, <int16> "}"

<powerpc code descriptor property> := "CodePowerPC" "{"
    <int32>, <int32> <optional name> "}"

<win32 x86 code property> := "CodeWin32X86" "{" <string> "}

<bitmap support> := "noBitmap" | "doesSupportBitmap"
<grayscale support> := "noGrayScale" | "doesSupportGrayScale"
<indexed support> := "noIndexedColor" | "doesSupportIndexedColor"
<RGB support> := "noRGBColor" | "doesSupportRGBColor"
<CMYK support> := "noCMYKColor" | "doesSupportCMYKColor"
<HSL support> := "noHSLColor" | "doesSupportHSLColor"
<HSB support> := "noHSBColor" | "doesSupportHSBColor"
<multichannel support> := "noMultichannel" | "doesSupportMultichannel"
<duotone support> := "noDuotone" | "doesSupportDuotone"
<LAB support> := "noLABColor" | "doesSupportLABColor"

<supported modes property> := "SupportedModes"
    "{"
    <bitmap support> ","
    <grayscale support> ","
    <indexed support> ","
    <RGB support> ","
    <CMYK support> ","
    <HSL support> ","
    <HSB support> ","
    <multichannel support> ","
    <duotone support> ","
    <LAB support>
    "}"

<filter case info property> := "FilterCaseInfo"
    "{"
        "{"
        <filter info case> # filterCaseFlatImageNoSelection
        <filter info case> # filterCaseFlatImageWithSelection
        <filter info case> # filterCaseFloatingSelection
        <filter info case> # filterCaseEditableTransparencyNoSelection
        <filter info case> # filterCaseEditableTransparencyWithSelection
        <filter info case> # filterCaseProtectedTransparencyNoSelection
        <filter info case> # filterCaseProtectedTransparencyWithSelection
        "}"
    "}"

<filter info case> :=
    <input matting> "," <output matting> ","
    <layer mask flag> "," <blank data flag> "," <copy source flag>

<input matting> :=
    "inCantFilter" |
    "inStraightData" |
    "inBlackMat" |
    "inGrayMat" |
    "inWhiteMat" |
    "inDefringe" |
    "inBlackZap" |
    "inGrayZap" |
```

```
    "inWhiteZap" |
    "inBackgroundZap" |
    "inForegroundZap"

<ouput matting> :=
    "outCantFilter" |
    "outStraightData" |
    "outBlackMat" |
    "outGrayMat" |
    "outWhiteMat" |
    "outFillMask"

<layer mask flag> := "doesNotFilterLayerMasks" | "filtersLayerMasks"
<blank data flag> := "doesNotWorkWithBlankData" | "worksWithBlankData"
<copy source flag> := "copySourceToDestination" |
    "doNotCopySourceToDestination"

<type creator pair> :=
    <OSType> "," <OSType>

<format file type property> :=
    "{"
    <type creator pair>
    "}"

<type creator pair list tail> :=
    <epsilon> |
    "," "{" <type creator pair> "}" <type creator pair list tail>

<type creator pair list> :=
    <epsilon> |
    "{" <type creator pair> "}" <type creator pair list tail>

<read types property> :=
    "{"
    <type creator pair list>
    "}"

<write types property> :=
    "{"
    <type creator pair list>
    "}"

<filtered types property> :=
    "{"
    <type creator pair list>
    "}"

<ostype list tail> :=
    <epsilon> |
    "," "{" <OSType> "}" <ostype list tail>

<ostype list> :=
    <epsilon> |
    "{" <OSType> "}" <ostype list tail>

<read extensions property> :=
    "{"
    <ostype list>
    "}"

<write extensions property> :=
    "{"
```

```
    <ostype list>
    "}"

<filtered extensions property> :=
    "{"
    <ostype list>
    "}"

<saves image resources flag> :=
    "fmtDoesNotSaveImageResources" | "fmtSavesImageResources"

<can read flag> :=
    "fmtCannotRead" | "fmtCanRead"

<can write flag> :=
    "fmtCannotWrite" | "fmtCanWrite"

<write if read flag> :=
    "fmtWritesAll" | "fmtCanWriteIfRead"

<format flags property> :=
    "{"
    <saves image resources flag> ","
    <can read flag> ","
    <can write flag> ","
    <write if read flag>
    "}"

<format maximum size property> :=
    "{"
    <int16>, <int16>
    "}"

<format maximum channels property> :=

<parsable types property> :=
    "{"
    <type creator pair list>
    "}"

<parsable extensions property> :=
    "{"
    <ostype list>
    "}"

<filtered parsable types property> :=
    "{"
    <type creator pair list>
    "}"

<filtered parsable extensions property> :=
    "{"
    <ostype list>
    "}"

<parsable clipboard types property> :=
    "{"
    <ostype list>
    "}"
```

File Extensions and Types

.ABR 117
.ACF 131
.ACM 122
.ACO 120
.ACT 119
.ADO 126
.AHS 128
.ALV 132
.AMP 116
.AMS 133
.API 130
.ASP 137
.ASR 135
.AST 138
.ASV 136
.ATF 140
.AXT 134
.CRV 124
.HSS 129

8BBR 117
8BCK 131
8BCO 120
8BCT 119
8BDT 126
8BFK 122
8BHA 129
8BHS 128
8BIC 130
8BLS 132
8BLT 116
8BMS 133
8BSC 124
8BSR 135
8BSS 137
8BST 138
8BSV 136
8BTF 140
8BXT 134

A

A5 register (680x0) 18
about boxes 15
acquire modules 55
    AcquireRecord parameter block 60
    acquireSelectorContinue 57
    acquireSelectorFinalize 58
    acquireSelectorFinish 57
    acquireSelectorPrepare 56
    acquireSelectorStart 57
AddPIResourceProc 31
Adobe Premiere 111
AdvanceStateProc 26
AllocateBufferProc 29
Apple MPW 3.3.1 18

B

BNDL resource 17
Boolean 13
Borland C++ 21

buffer suite 29
BufferSpaceProc 30

C

callback suites 24
Callback suites description 28
callbacks
    AddPIResourceProc 31
    AdvanceStateProc 26
    AllocateBufferProc 29
    BufferSpaceProc 30
    ColorServicesProc 26
    CountPIResourcesProc 31
    DeletePIResourceProc 31
    DisplayPixelsProc 25
    DisposePIHandleProc 32
    FreeBufferProc 30
    GetPIHandleSizeProc 32
    GetPIResourceProc 31
    GetPropertyProc 36
    HostProc 27
    LockBufferProc 29
    LockPIHandleProc 32
    NewPIHandleProc 32
    PIResampleProc 34
    ProcessEventProc 25
    RecoverSpaceProc 33
    SetPIHandleSizeProc 32
    SetPropertyProc 36
    SpaceProc 27
    TestAbortProc 25
    UnlockBufferProc 30
    UnlockPIHandleProc 33
    UpdateProgressProc 25
cfrg resource 19
CNVTPIPL.EXE 21, 22, 23, 39
code fragment manager (Macintosh) 17
Code68K 19
CodePowerPC 19
CodeWarrior, *See* Metrowerks
color mode data 104
ColorServicesInfo 145
ColorServicesProc 26
complexProperty 36
CountPIResourcesProc 31

D

DeletePIResourceProc 31
DialogUtilities 18
direct callbacks 24
DisplayPixelsProc 25
DisposePIHandleProc 32
    disposing complex properties 36
Dissolve 72
DummyExport 65
DummyScan 55

E

encapsulated PostScript files 110
error reporting 14
export modules 65

ExportRecord parameter block 68
exportSelectorContinue 67
exportSelectorFinish 67
exportSelectorPrepare 66
exportSelectorStart 66
extension modules 12

F

fat plug–ins (Macintosh) 17
filmstrip files 111
filter modules 72
FilterRecord parameter block 76
filterSelectorContinue 74
filterSelectorFinish 75
filterSelectorParameters 73
filterSelectorPrepare 74
filterSelectorStart 74
FilterCaseInfo 48
FlagSet 13
format modules 83
estimate sequence 89
file filtering 84
FormatRecord parameter block 92
formatSelectorEstimateContinue 89
formatSelectorEstimateFinish 89
formatSelectorEstimatePrepare 89
formatSelectorEstimateStart 89
formatSelectorOptionsContinue 88
formatSelectorOptionsFinish 88
formatSelectorOptionsPrepare 88
formatSelectorOptionsStart 88
formatSelectorReadContinue 86
formatSelectorReadFinish 86
formatSelectorReadPrepare 85
formatSelectorReadStart 85
formatSelectorWriteContinue 91
formatSelectorWriteFinish 91
formatSelectorWritePrepare 90
formatSelectorWriteStart 90
options sequence 88
read sequence 85
write sequence 90
writing a file 87
FreeBufferProc 30

G

GetPIHandleSizeProc 32
GetPIResourceProc 31
getPropertyObsolete 36
GetPropertyProc 36

H

handle suite 32
hardware accelerator modules 12
HistoryExport 23, 65
HostProc 27

I

IllustratorExport 23
image resources 91, 104
image services suite 34

interpolate1D 34
interpolate2D 34

L

load files
    Arbitrary Map 116
    Brushes 117
    Color Table 119
    Colors 120
    Command Settings File 122
    Curves 124
    Custom kernel 131
    Duotone options 126
    Halftone screens 128
    Hue/Saturation 129
    Ink colors setup 130
    Levels 132
    Monitor setup 133
    Replace color/Color range 134
    Scratch Area 135
    Selective color 136
    Separation setup 137
    Separation tables 138
    Transfer function 140
load files, description 115
LockBufferProc 29
LockPIHandleProc 32

M

Macintosh
    cfrg resource 19
    code fragment manager 17
    creating plug–ins 17
    fat plug–ins 17
    PowerMac native plug–ins 17
MACTODOS.EXE 22
memory management strategies 16
    setting maxData 16
Metrowerks CodeWarrior 18
    notes for CodeWarrior Bronze users 19
    notes for CodeWarrior Gold users 19
Microsoft Windows 21

N

NewPIHandleProc 32

O

OSType 13

P

parser modules 12
Paths to Illustrator 65
PHOTOSHO.INI 12, 22
Photoshop EPS files 110
PICFMCodeDesc 44
PiMI resources 17, 53
PiPL 17, 19
    PI68KCodeProperty 44
    PI68KFPUCodeProperty 44
    PICategoryProperty 43

PIExpFlagsProperty 46
PIFilterCaseInfoProperty 47
PIFilteredExtProperty 50
PIFilteredTypesProperty 50
PIFmtFileTypeProperty 50
PIFmtFlagsProperty 51
PIFmtMaxChannelsProperty 51
PIFmtMaxSizeProperty 51
PIImageModesProperty 43
PIKindProperty 42
PINameProperty 43
PIPowerPCCodeProperty 44
PIPriorityProperty 42
PIProperty structure 40
PIReadExtProperty 50
PIReadTypesProperty 50
PIRequiredHostProperty 43
PIVersionProperty 42
PIWin32X86CodeProperty 45
plug–in property list structure 40
PiPL grammar 151
PIPropertyList 40
PIResampleProc 34
PIUtilities 18, 22
PIWin32X86CodeDesc 45
PIWin32X86CodeProperty 21
plug–in file types and extensions 13
plug–in hosts 10
plug–in modules 10
Plug–in Property List *See* PiPL
pluginData 14
PLUGINDIRECTORY 12, 22
PlugInInfo (PiMI) 53
PlugInMonitor 148
pluginParamBlock 14
Premiere 111
ProcessEventProc 25
property keys (property suite) 37
property suite 36
propBigNudgeH 37
propBigNudgeV 37
propCaption 38
propChannelName 37
propClippingPathIndex 37
propHardwareGammaTable 38
propImageMode 37
propInterpolationMethod 37
propNumberOfChannels 37
propNumberOfPaths 37
propPathContents 37
propPathName 37
propRulerUnits 37
propSerialString 38
propTargetPathIndex 37
propWorkPathIndex 37
pseudo–resource suite 31
PSImagePlane structure 34
PSPixelMap 142
PSPixelMask 144
PString 13

R

RecoverSpaceProc 33
resource
    cfrg 19
Resources 17
    vers resource 17

S

segmentation (680x0) 18
SetPIHandleSizeProc 32
SetPropertyProc 36
simpleProperty 36
SpaceProc 27
Symantec C++ (Windows) 21

T

TestAbortProc 25
TIFF files 113
TypeCreatorPair 13

U

UnlockBufferProc 30
UnlockPIHandleProc 33
UpdateProgressProc 25

V

Version 2.5 11
Visual C++ 21
VPoint 13
VRect 13

W

Windows 21
WinUtils 22
Writing 89