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 ships with several file format modules including GIF, MacPaint, and BMP.

Import and Export modules may also be used to read and write files. You should create a Format module if you want your users to treat your files in the same fashion as native Photoshop files. Use a Format module if:

1. 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.

2. You want users to be able to double–click a document to launch Photoshop or associate your file extension with the Photoshop application.

You may *not* want to use a Format module if:

1. With respect to Photoshop, your file format is read–only or write–only.

2. The image compression and/or color space conversion on multiple reading and writing would result in unacceptable image degradation.

Table 8–1: Format file types

OS	Filetype/extension
Mac OS	`8BIF`
Windows	`.8BI`

SampleCode/Format/SimpleFormat

SimpleFormat is a sample Format module. This module is written to use the `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:

1. *formatSelectorFilterFile* 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.

2. The *read* sequence is used to read image files.

Writing a file consists of three sequences:

1. 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.

2. The *estimate* sequence estimates the file size so that the host can decide whether there is enough disk space available.

3. The *write* sequence actually writes the file.

Note: Your plug–in should validate the contents of its globals and parameters whenever it starts processing if there is a danger of it crashing from bad parameters.

Globals and scripting
The scripting system passes its parameters at every selector call. While it is possible to use the scripting system to store all your parameters, for backwards compatibility, it is recommended you track your parameters with your own globals. Once your globals are initialized, you should read your scripting-passed parameters and override your globals with them.

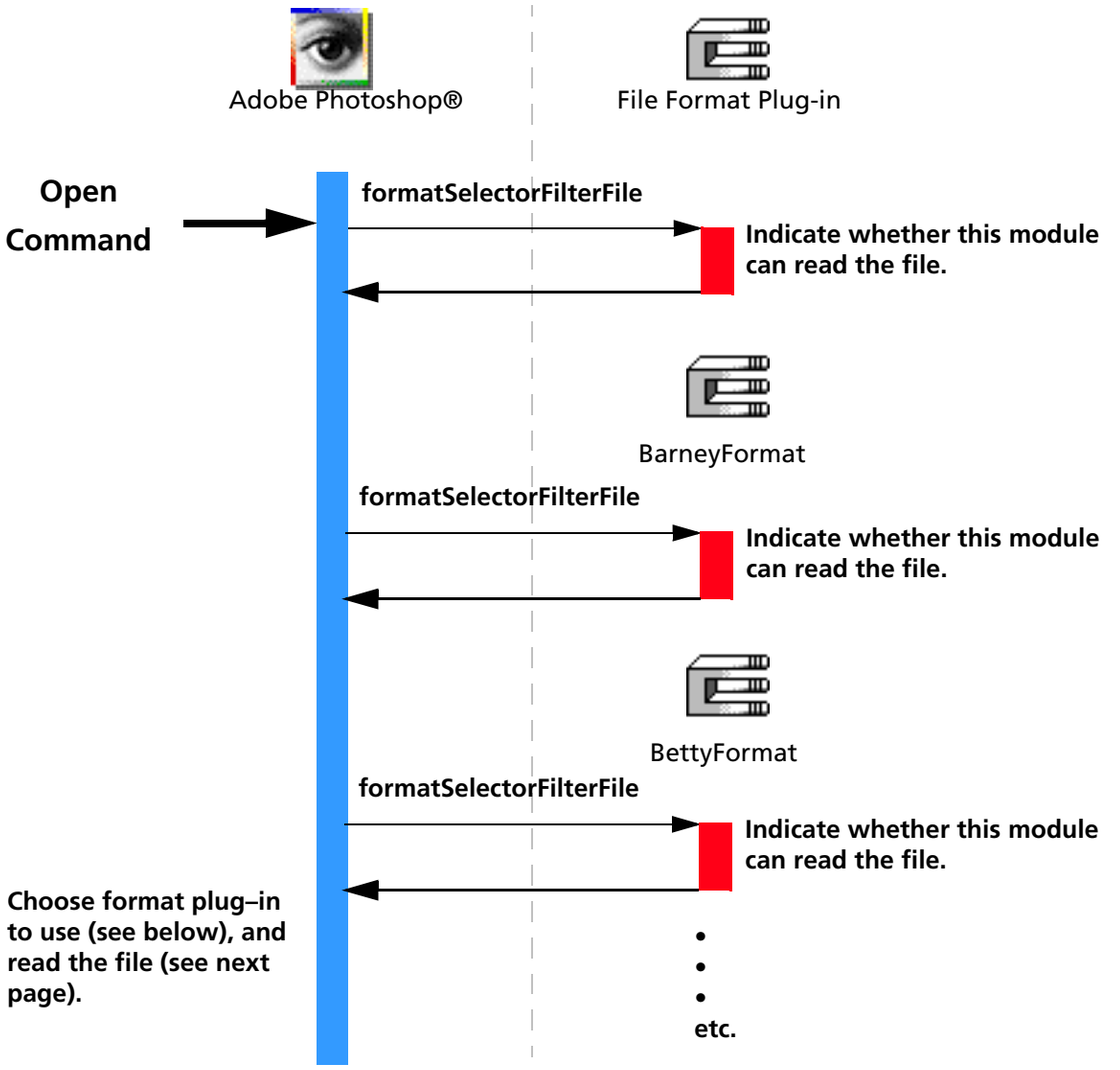
The most effective way to do this is:

1. First call a *ValidateMyParameters* routine to validate (and initialize if necessary) your global parameters.

2. Then call a *ReadScriptingParameters* routine to read the scripting parameters and then write them to your global structure.

This way, the scripting system overrides your parameters, but you can use the initial values if the scripting system is unavailable or has parameter errors, and you can use your globals to pass between your functions.

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 modules that list the Mac OS file type or Windows file extension as a supported format. For each such plug–in module, Photoshop will call the plug–in with `formatSelectorFilterFile`. 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;
```

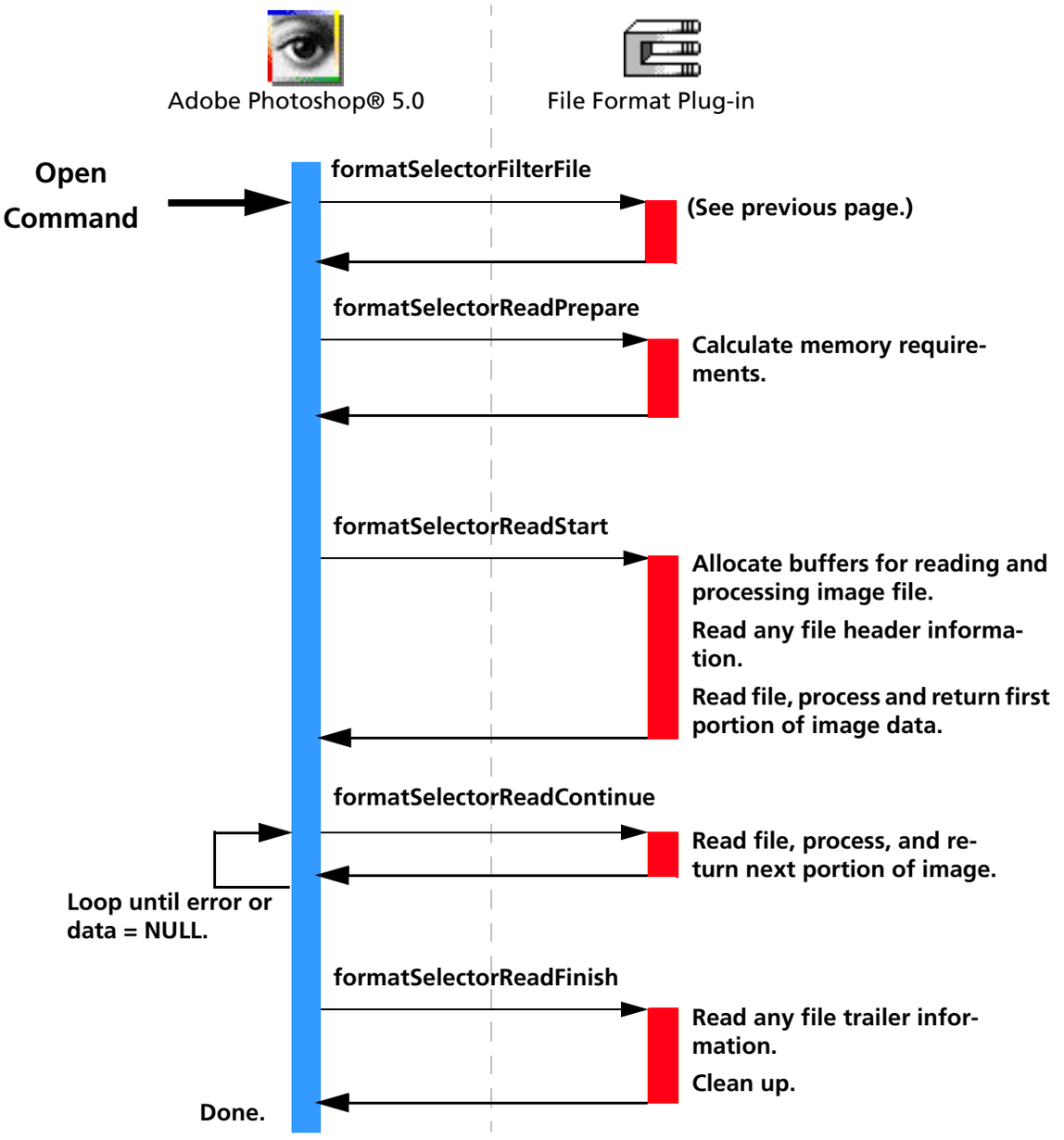
If more than one format module can read the file, Photoshop uses the following priority scheme to determine which plug–in module to use:

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.

Scripting at formatSelectorReadStart

If you are supporting scripting, read any scripting parameters here to override any default parameters. The scripting system will also return whether to show your dialog or not.

imageMode & imageSize
You should initialize *imageMode*, *imageSize*, *depth*, *planes*, *imageHRes* and *imageVRes*.

Reading an indexed color image (redLut, greenLut & blueLUT)
If an indexed color image is being opened, you should also set *redLUT*, *greenLUT* and *blueLUT*.

imageRsrcData
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. For more information about Photoshop image resources, see *Photoshop File Formats.pdf*.

theRect, loPlane & hiPlane
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*.

data, colBytes, rowBytes, planeBytes & planeMap
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. For example, to read just the red plane of an RGB color image, use the parameter values in Table 8–2.

Table 8–2: Read red plane of RGB

Parameter	Value
loPlane	0
hiPlane	0
colBytes	1
rowBytes	width of the area being read
planeBytes	ignored, since `loPlane=hiPlane`.

If you wish to read the RGB data in interleaved form (RGBRGB...), use the values shown in Table 8–3.

Table 8–3: Read RGB data in interleaved form

Parameter	Value
loPlane	0
hiPlane	2
colBytes	3
rowBytes	3 * width of the area being read
planeBytes	1

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=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 `formatSelectorReadContinue` 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.

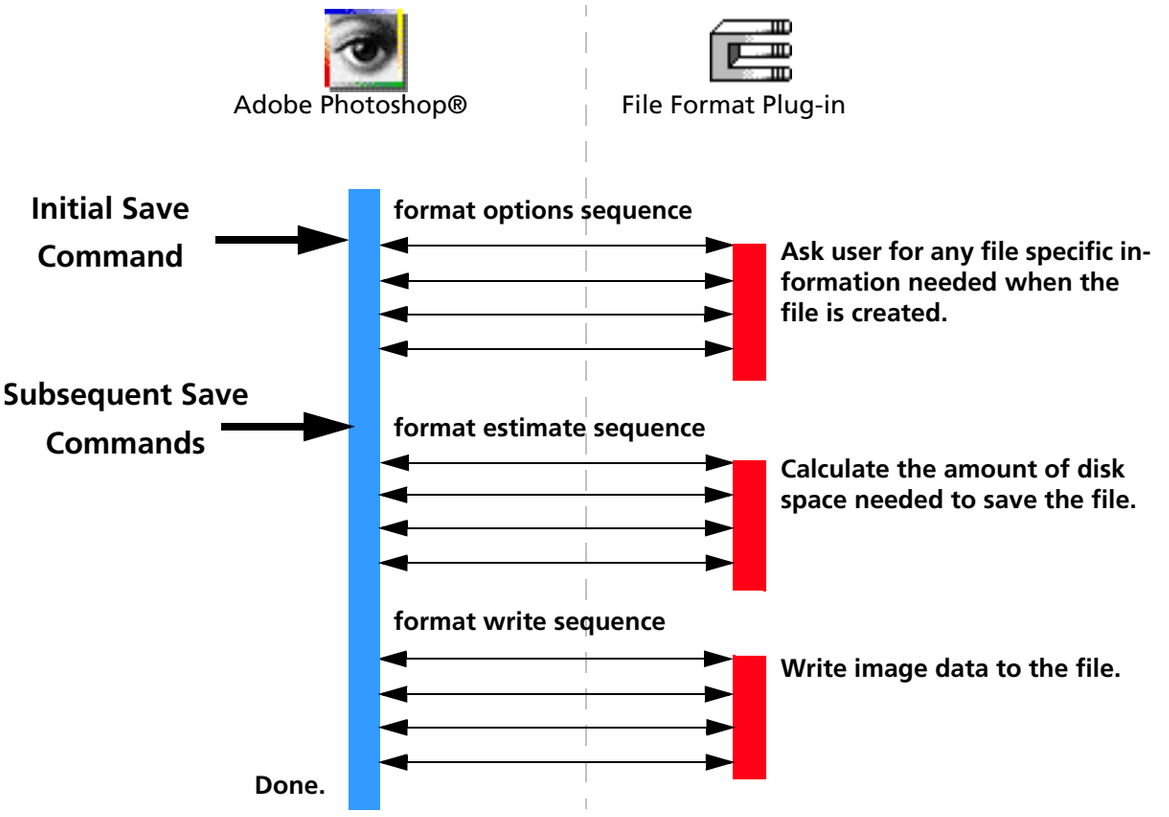
If Photoshop detects *Command–period* in the Mac OS or *Escape* in Windows while processing the results of a `formatSelectorReadContinue` call, it will call `formatSelectorReadFinish`.

Note: Be careful processing user-cancel events during `formatSelectorReadContinue`. Normally your plug-in would be expecting another `formatSelectorReadContinue` call. If the user cancels, the next call will be `formatSelectorReadFinish`, *not* `formatSelectorReadContinue`!

Scripting at formatSelectorReadFinish

If your plug-in is scripting-aware and you've changed any initial parameters, you should pass a complete descriptor back to the scripting system in the `PIDescriptorParameters` structure.

Writing a file



Writing a file involves either two or three distinct sequences, each similar in structure. The details of these sequences are described on the following pages.

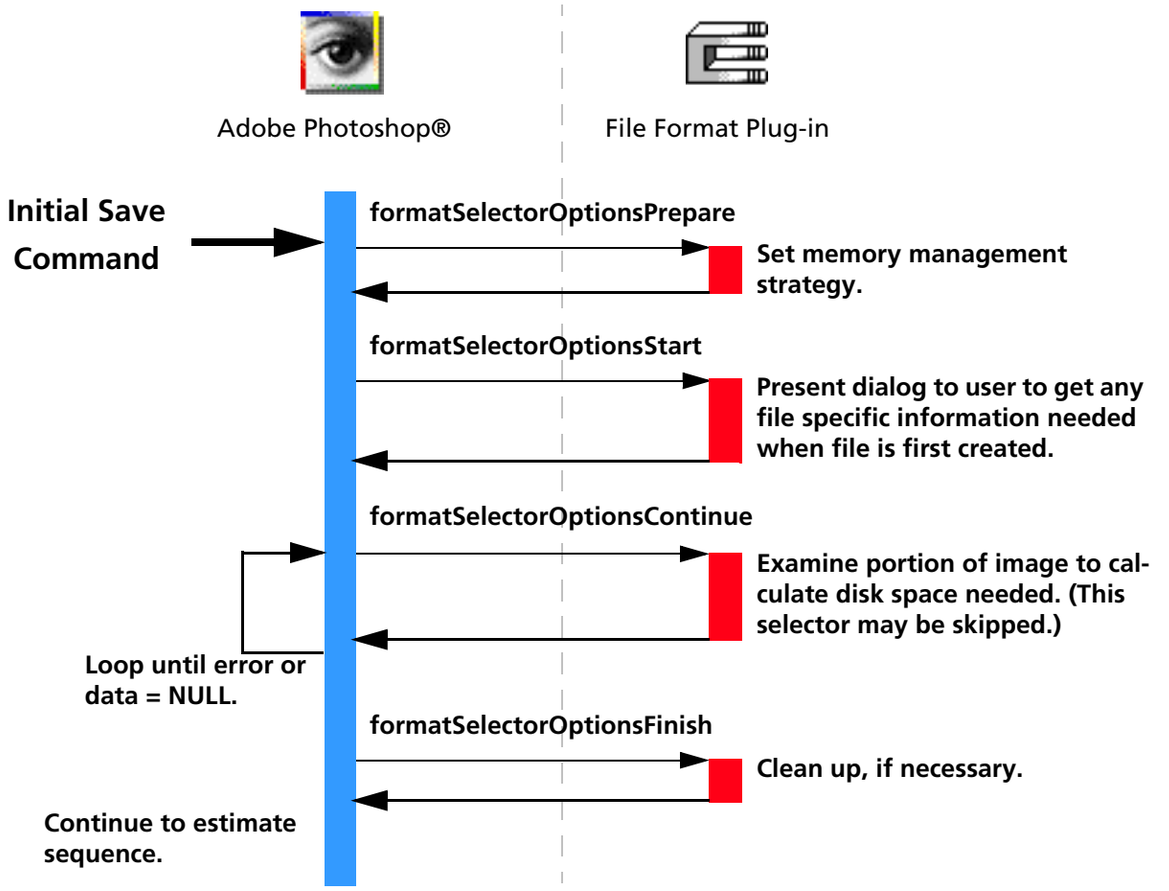
When a document is first saved, Photoshop calls your Format plug–in in this order:

1. the *options* sequence,
2. the *estimate* sequence,
3. the *write* sequence.

After a document has been saved once, each time the user saves the file again, the plug-in is called without the options sequence:

1. the *estimate* sequence,
2. the *write* sequence.

Writing a file (options sequence)



formatSelectorOptionsPrepare

formatSelectorOptionsPrepare 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

formatSelectorOptionsStart 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.

Scripting at formatSelectorOptionsStart

If you are supporting scripting, read any scripting parameters here to override any default parameters. The scripting system will also return whether to show your dialog or not.

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.

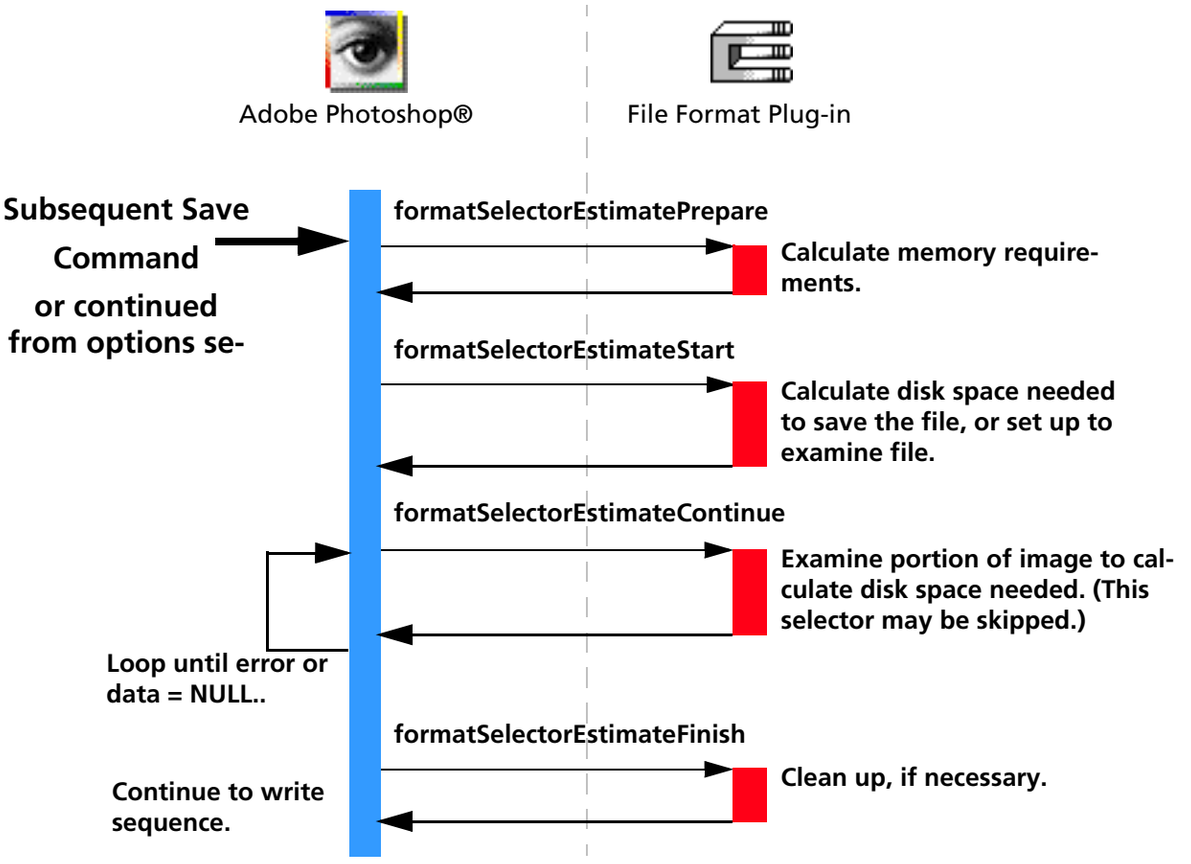
formatSelectorOptionsContinue

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

formatSelectorWritePrepare 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=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

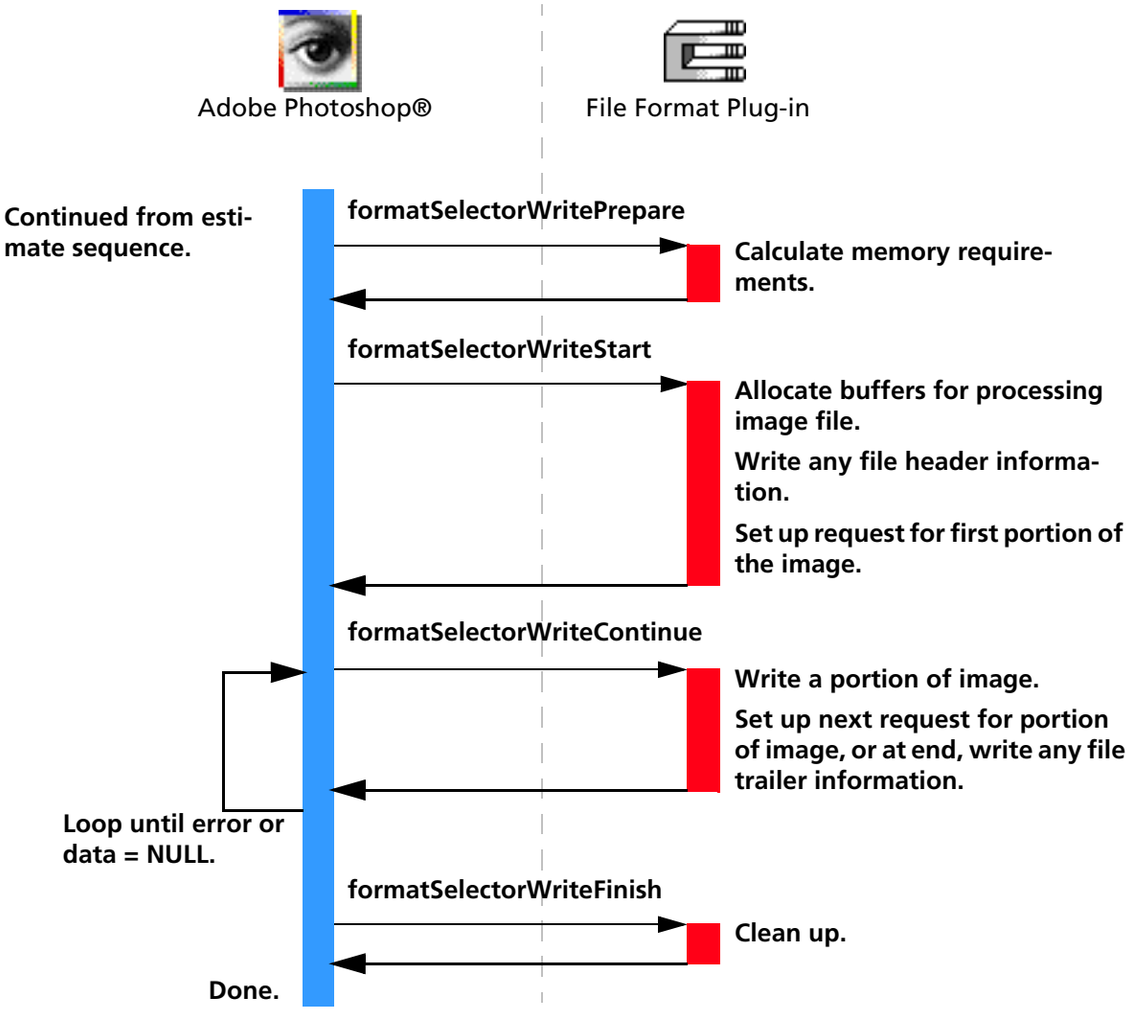
If the `data` field in the `FormatRecord` is set to `NULL` in the `formatSelectorEstimateStart` call, this selector will not be called at all. Otherwise, your plug–in can request parts of the image from which you can compute the minimum and maximum bytes to store the file. Refer to `formatSelectorWriteStart`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

formatSelectorWritePrepare 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.

theRect, loPlane & hiPlane

The area of the image being requested from the plug–in host is specified by *theRect*, *loPlane*, and *hiPlane*.

data

The actual pixel data is pointed by *data*.

colBytes, rowBytes, planeBytes & planeMap

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. For example, to return just the red plane of an RGB color image, use the parameter values in Table 7-3.

Table 8–4: Return red plane of RGB

Parameter	Value
loPlane	0
hiPlane	0
colBytes	1
rowBytes	width of the area being returned
planeBytes	ignored, since `loPlane=hiPlane`.

If you wish to return the RGB data in interleaved form (RGBRGB...), use the values shown in Table 7-4.

Table 8–5: Return RGB data in interleaved form

Parameter	Value
loPlane	0
hiPlane	2
colBytes	3
rowBytes	3 * width of the area being read
planeBytes	1

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

formatSelectorWriteFinish 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* in the Mac OS or *Escape* in Windows while processing the results of a `formatSelectorWriteContinue` call, it will call the `formatSelectorWriteFinish` routine.

Note: Be careful processing user-cancel events during `formatSelectorWriteContinue`. Normally your plug-in would be expecting another `formatSelectorWriteContinue` call. If the user cancels, the next call will be `formatSelectorWriteFinish`, *not* `formatSelectorWriteContinue`!

Scripting at formatSelectorWriteFinish
If your plug-in is scripting-aware and you've changed any initial parameters, you should pass a complete descriptor back to the scripting system in the `PIDescriptorParameters` structure.

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. These errors and more detail is available in `PIFormat.h`.

```
#define formatBadParameters     -30500    // an error with the interface
#define formatCannotRead        -30501    // no scanner installed
```

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 selectors will be called: options, estimate, nor write.

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 8–6: 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.
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. In Windows, this is the file handle of the file returned by `OpenFile()`.
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. In Windows, this field is undefined.
FSSpec *	fileSpec	Full file specification.
int16	imageMode	The `formatSelectorReadStart` routine should set this field to inform Photoshop what mode image is being imported (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 8–6: 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 and Import 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.
int16	hiPlane	

Table 8–6: FormatRecord structure (Continued)

Type	Field	Description
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.
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]=3`, `planeMap[1]=0`, `planeMap[2]=1`, and `planeMap[3]=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=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 is 1, LSB = bit 0, the mode is supported. This field can be used by plug–ins to disable reading unsupported file formats.

Table 8–6: 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. This field is checked after each call to `formatSelectorRead` and `formatSelectorContinue` and as soon as it is not `NULL` Photoshop parses the handle as a block of image resource data for the current document.
int32	imageRsrcSize	This is the size of the handle `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.
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 8–6: 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 since 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. See chapter 3.
These fields are new since version 3.0.4 of Adobe Photoshop.		
PropertyProcs *	propertyProcs	A pointer to the Property callback suite. See chapter 3.
int16	tileWidth	The host reports the width and height of a tile, which would be the best unit to work in, if possible.
int16	tileHeight	
int16	tileOrigin	The origin of the tiling system.
These fields are new since version 4.0 of Adobe Photoshop.		
PIDescriptorParameters *	descriptorParameters	Descriptor callback suite. See chapter 3.
Str255 *	errorString	If you return with result=`errReportString` then whatever string you store here will be displayed as: "Cannot complete operation because *string*".
These fields are new since version 5.0 of Adobe Photoshop.		
int32	maxValue	Used for read with 16bit depth only.
SPBasicSuite *	sSPBasic	PICA basic suite.
void *	plugInRef	Plugin reference used by PICA.
int32	transparentIndex	If IndexedColor, and between 0 and 256, this is the index of the transparent color (for GIF).
Handle	iCCprofileData	Handle containing the ICC profile for the image. (NULL if none.) For reads: The handle must be allocated using Photoshop's handle suite; Photoshop will use the data after the Finish call and Photoshop will free the handle. For writes: Photoshop allocates the handle using Photoshop's handle suite; The handle is unlocked while calling the plug-in; The handle will be valid from Start to Finish; Photoshop will free the handle after Finish
int32	iCCprofileSize	Size of profile.
int32	canUseICCProfiles	Non-zero if the host can accept/export ICC profiles if this is zero, you'd better not set or dereference iCCprofileData.

Table 8–6: FormatRecord structure (Continued)

Type	Field	Description
These fields are new since version 5.5 of Adobe Photoshop.		
int32	lutCount	Number of entries in the indexed color table. This should include the transparent index if any. Plug-ins should pad out the color table with black for backword compatibility.
These fields are new since version 6.0 of Adobe Photoshop.		
int32	preferredColor-Modes	A bitmask indicating which color modes are preferred by the host. This will be zero if not set. The plug-in is free to ignore the hint (though it is wise to obey the hostModes information). Note that this is an int32 rather than an int16 to allow for future expansion.
int32	convertMode	0 if the host does not support conversions. -1 if the host does support conversions after reading. The plug-in can set this field to the index of a color mode and this will be treated by the host as a request to automatically convert to that mode. Hosts can ignore this request but they will make the host+plug-in combination appear broken.
VPoint	preferredSize	0,0 if not set. This is a hint about how big an image the wants. Note that this field uses 32-bit coordinates rather than 16-bit coordinates. At some point the rest of the API will probably be extended to handle big coordinates and it seemed reasonable to make the hint big enough to be ready for that so that we don't need to adjust it as well.
int32	imageIndex	0 if not set. One means first image, etc. Only used for PDF Image extractor currently.
int32	transparency-Plane	When writing files: Set to zero if host does not support transparency or there is no transparency. Set to the index of the plane containing transparency information if the host does support transparency. Any alpha channels after this index are pushed down by one. Note that the interpretation of zero is chosen for compatibility with old hosts, but it also means that the transparency plane will never be the first plane. When reading files: This is set to zero by hosts that do not support transparency. This is set to -1 by hosts that support transparency. The plug-in should set it to the index of the plane containing transparency information if there is one.
int32	transparency-Matting	0: no matte, 1: black matte, 2: gray matte, 3: white matte When reading files, how should the transparency data be interpreted with respect to color matting. When writing, this is set by the plug-in to indicate whether the host should be premultiplied with a choosen color. The default value is no matting (unassociated).
These fields are new since version 7.0 of Adobe Photoshop.		
ChannelPortProcs *	channelPortProcs	Suite for passing pixels through channel ports.
ReadImageDocu-mentDesc *	documentInfo	The document info for the document being written.
Boolean	openForPreview	Makes the smallest previewable size if it can.
These fields are new since version CS (8.0) of Adobe Photoshop.		

Table 8–6: FormatRecord structure (Continued)

Type	Field	Description
int32	browserRotation	When reading files, this field lets the plug-in know what rotation and/or flip will be applied to the image after opening. This lets the plug-in display any preview images correctly. Lowest two bits are number of 90 degree CW rotations. The PIFmtBrowserRotationFlipFlag bit is a flip horizontal flag. The PIFmtBrowserRotationDisableFlag bit lets the plug-in know the host does not support this field. The PIFmtBrowserRotationEnableFlag bits lets the plug-in know this field is read-write. This field is also be used when requesting a value for the PIFmtPRotate property and when calling the formatSelectorLosslessRotate selector.
int32	HostSupports32BitCoordinates	set by host if the host supports 32 bit plugin API
int32	PluginUsing32BitCoordinates	set to nonzero by the plugin if it is using the 32 bit fields
VPoint	imageSize32	Size of image in 32 bit coordinates replaces imageSize
VRect	theRect32	Rectangle being returned in 32 bit coordinates replaces theRect
uint32	requestedFileProperty	Set to a valid format file property to request the value for that property.
uint32	filePropertyValue	The requested property value is returned in this field.
uint32	fileCount	Count of files for bulk settings selector.
Handle	xmp	XMP handle for read and write, only valid when using formatSelectorXMPRead and formatSelectorXMPWrite. NOTE: Use the propertyProcs and propXMP to read and write the XMP data during normal reads and writes. For reads: The handle must be allocated using Photoshop's handle suite. Photoshop will use the data after the call and Photoshop will free the handle. For writes: Photoshop allocates the handle using Photoshop's handle suite. The handle is unlocked while calling the plug-in .The handle will be valid only during formatSelectorXMPWrite Photoshop will free the handle
int32	supportsSkipFile	Is the formatSkipFile error code supported? This error code tells the host to skip this file when processing multiple files.
char[88]	reserved	Reserved for future use. Set to zero.