Filters and Reusable Streams

Adobe® Developers Association

9 October 1997

Technical Note #5603

LanguageLevel 3

Adobe Systems Incorporated

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

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

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

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

PN LPS5603

Adobe Systems Incorporated

Copyright © 1997 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated.

No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Adobe, PostScript, PostScript 3, and the PostScript logo are trademarks of Adobe Systems Incorporated. Apple and Macintosh are trademarks of Apple Computer, Inc. registered in the U.S. and other countries. All other trademarks are the property of their respective owners.

Contents

1 Filters and Reusable Streams 11
- Overview of Filters 11
- General Changes to PostScript Filters 12
- The Benefits of Using Filters and Reusable Streams 14

2 ASCII-Based Filters 15
- The ASCIIHexEncode and ASCIIHexDecode Filters 15
- The ASCII85Encode and ASCII85Decode Filters 17

3 LZW Filters 18

4 RunLength Filters 19

5 CCITTFax Filters 20

6 NullEncode Filter 20

7 DCT Filters 21

8 SubFileDecode Filter 23

9 New Filters for LanguageLevel 3 24
- FlateEncode and FlateDecode Filters 24
- Reusable Streams and the ReusableStreamDecode Filter 28
- GIFDecode Filter 35
- PNGDecode Filter 36

Adobe Systems Incorporated

Adobe Systems Incorporated

Tables

Table 1 Filters Available in the PostScript Language 12
Table 2 Keys Available in All Encode Filter Dictionaries 13
Table 3 Keys Available in All Decode Filter Dictionaries 13
Table 4 Keys in the LZWDecode Filter Dictionary 18
Table 5 Keys in the FlateEncode Filter Dictionary 25
Table 6 Keys in the FlateDecode Filter Dictionary 26
Table 7 Keys in the ReusableStreamDecode Dictionary 31

Adobe Systems Incorporated

Adobe Systems Incorporated

Examples

Example 1 Use of the ASCIIHexEncode Filter 16
Example 2 Use of an ASCII85Decode Filter to Decode a PostScript Language Stream 17
Example 3 Use of the ASCII85Decode and RunLengthDecode Filters 19
Example 4 Use of the DCTEncode Filter 21
Example 5 Decoding and Printing a JPEG-compressed File 22
Example 6 Use of the SubFileDecode Filter 23
Example 7 Use of the ReusableStreamDecode Filter 33

Adobe Systems Incorporated

Preface

This Document

This is the original release for *Filters and Reusable Streams*, a document that provides a detailed description of the LanguageLevel 3 extensions to filters.

Intended Audience

This document is written for software developers who are interested in learning about filters and reusable streams or adding these capabilities to an application that supports PostScript® display or printing devices.

It is assumed that the developer is already familiar with how filters work in previous levels of the PostScript language.

Organization of This Document

Section 1, “Filters and Reusable Streams,” gives a general overview of filters in the PostScript language. It also covers some of the changes to filters for LanguageLevel 3. Finally, it presents some of the major benefits of using filters and reusable streams in applications.

Section 2, “ASCII-Based Filters,” reviews the ASCIIHex and ASCII85 filters.

Section 3, “LZW Filters,”discusses the changes to the LZWEncode and LZWDecode filters.

Section 4, “RunLength Filters,” reviews the RunLengthEncode and RunLengthDecode filters.

Section 5, “CCITTFax Filters,” covers the changes to the filters available for FAX data.

Section 6, “NullEncode Filter,” reviews this special encoding filter.

Section 7, “DCT Filters,” reviews the filters used for JPEG files.

Section 8, “SubFileDecode Filter,” presents the changes to this decode filter.

Section 9, “New Filters for LanguageLevel 3,” discusses the filters that have been introduced for LanguageLevel 3, including the FlateEncode and FlateDecode filters, the GIFDecode filter, the PNGDecode filter, and the ReusableStreamDecode filter.

Related Publications

Supplement: PostScript Language Reference Manual (LanguageLevel 3 Specification and Adobe PostScript 3™ Version 3010 Supplement), available from the Adobe Developers Association, describes the formal extensions to the PostScript language that have occurred since the publication of the PostScript Language Reference Manual, Second Edition. This supplement also includes all LanguageLevel 3 extensions available in version 3010.

PostScript Language Reference Manual, Second Edition (Reading, MA: Addison-Wesley, 1991) is the developer’s reference manual for the PostScript language. It describes the syntax and semantics of the language, the imaging model, and the effects of the graphical operators.

C language source code is available for some filters from the Adobe Developers Association. Although these files were written to accompany the release of an earlier level of the PostScript language, they should, for the most part, apply to discussions in this document.

The Bibliography lists some of the many outside sources of information on filters and filter specifications.

Statement of Liability

THIS PUBLICATION AND THE INFORMATION HEREIN IS FURNISHED AS IS, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY ADOBE SYSTEMS INCORPORATED. ADOBE SYSTEMS INCORPORATED ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES, MAKES NO WARRANTIES OF ANY KIND (EXPRESS, IMPLIED, OR STATUTORY) WITH RESPECT TO THIS PUBLICATION, AND EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR PARTICULAR PURPOSES, AND NONINFRINGMENT OF THIRD-PARTY RIGHTS.

Adobe Systems Incorporated

Filters and Reusable Streams

1 Filters and Reusable Streams

1.1 Overview of Filters

Many PostScript language operators and programs now produce or consume data streams that can be encoded in various forms, usually for purposes of compression and/or for reliable transmission through seven-bit ASCII networks. Such data sources are defined in terms of filters that perform some kind of transformation on an incoming or outgoing data stream.

A filter is in fact a special kind of file object. The semantics of filters expand the definition of files in the following way: the data target or data source of a filter can be a string or a procedure, not just a file such as **currentfile** or a disk file. In the case of a string, the filter simply writes bytes to, or reads bytes from, the string.

Filters can be cascaded; that is, a source of data can be decoded, for instance, in ASCII85 format (see Section 2.2) and then further decompressed through a Flate filter (see Section 9.1). Example 3 shows how two filters might be used together.

Filters were introduced in an earlier level of the PostScript language and their capabilities have now been expanded in LanguageLevel 3. The changes to filters are covered in the next section.

Note *For more information on data compression using PostScript language filters, see Technical Note #5115, "Supporting Data Compression in PostScript Level 2 and the filter Operator."*

Adobe Systems Incorporated

1.2 General Changes to PostScript Filters

Some new filter types have been added to LanguageLevel 3; in addition, several changes have been made to some of the filters already supported in the PostScript language. Although some of the filters have not changed at all for this release of the PostScript language, they are reviewed in this document for completeness.

Table 1 illustrates the various types of filters available in the PostScript language and when they were introduced.

Table 1 *Filters Available in the PostScript Language*

Encode Filters	Decode Filters	LanguageLevel
ASCIIHexEncode	**ASCIIHexDecode**	2
ASCII85Encode	**ASCII85Decode**	2
LZWEncode	**LZWDecode**	2
RunLengthEncode	**RunLengthDecode**	2
CCITTFaxEncode	**CCITTFaxDecode**	2
DCTEncode	**DCTDecode**	2
	SubFileDecode	2
NullEncode		2
FlateEncode	**FlateDecode**	3
	GIFDecode	3
	Reusable-StreamDecode	3
	PNGDecode	3

In LanguageLevel 3, all encoding filters, with the exception of the **NullEncode** filter, have become optional in PostScript printers. All of the decoding filters, except for **GIFDecode** and **PNGDecode**, are still required. The **GIFDecode** and **PNGDecode** filters are considered optional filters used only as part of the implementation of web printing. Standard PostScript language programs should not invoke these two decode filters. The **resourceforall** or **resourcestatus** operators should be used to determine the list of available filters in a given device. This list of filters can be found in the implicit resource category called **Filter**.

Note *To ensure portability, PostScript language programs that are page descriptions should not invoke the optional encode filters.*

As of LanguageLevel 3, all encode and decode filters now take an optional dictionary of one or more filter parameters (To be more exact, dictionary support for filters was added into LanguageLevel 2 after the *PostScript Language Reference Manual, Second Edition* was published). Parameters available to all filter dictionaries are shown in Table 2 and 3 and described below. Dictionary parameters for specific filters, if available, are discussed in Sections 2 through 9.

Table 2 *Keys Available in All Encode Filter Dictionaries*

Key	Type	
CloseTarget	Boolean	optional

CloseTarget for encode filters has been added as an optional Boolean key in the optional encode filter dictionary. If this key is missing, its value defaults to false. If the value of **CloseTarget** is true, then whenever the filter is closed, either explicitly by the **closefile** operator or implicitly (by the **restore** operator, garbage collection, or reaching end of data (EOD)), then, if applicable, the data target of the filter will also be closed; this may be an iterative process. If the value of the **CloseTarget** key is false, no additional action is taken on the data source or target (this is the default behavior of LanguageLevel 2 devices).

Table 3 *Keys Available in All Decode Filter Dictionaries*

Key	Type	
CloseSource	Boolean	optional

CloseSource for decode filters has been added as an optional Boolean key in the optional decode filter dictionary. Its definition and use is the same as that for the **CloseTarget** key, described above.

Note *Several new instances of the implicit resource category **Filter** have been added in LanguageLevel 3. These new instances are **GifDecode**, **PNGDecode**, **FlateEncode**, **FlateDecode**, and **ReusableStreamDecode**. For more information, see Section 3.3 of the Supplement: PostScript Language Reference Manual.*

Note *There are five new instances of the implicit resource **Filter**. These are **FlateEncode**, **FlateDecode**, **GIFDecode**, **PNGDecode**, and **ReusableStreamDecode**. See Section 3.1 of the Supplement: PostScript Language Reference Manual, for more information.*

Adobe Systems Incorporated

Adobe Systems Incorporated

1.3 The Benefits of Using Filters and Reusable Streams

The following benefits can be achieved from using filters and reusable streams:

- The **GIFDecode** and **PNGDecode** filters enable the PostScript interpreter to directly extract the image data from GIF format and PNG format images, respectively. GIF is an image format commonly used for web content. It is based on LZW compression. PNG is a proposed new standard, based on Flate compression. Since Flate is an open standard, it can be assumed that PNG will become one of the standard image formats for web content.

  In addition to decompressing the image content, the **GIFDecode** and **PNGDecode** filters also strip off all the header information, which is more than is done by the LZW and Flate filters.

- Flate filters discover and exploit many patterns in input data, whether it be images or text. Because of their cascaded, adaptive Huffman coding, Flate-encoded output is usually substantially more compact (tighter compression) than LZW-encoded output given the same input.

- The **FlateDecode** filter can be used to decompress raster images in the Portable Network Graphics (PNG) format. The PNG format is an extensible file format that provides lossless, portable, and well-compressed storage of raster images. The DEFLATE compressed data streams within PNG are stored in the zlib format. PNG provides a patent-free replacement for GIF and can also replace many common uses of TIFF.

2 ASCII-Based Filters

The ASCII-based filters include the ASCIIHex format and the ASCII85 format. They can encode binary data in either hexadecimal or base-85 format, and decode these formats back to a standard binary format. They both use special character sequences to mean end of data (EOD). Binary based filters usually recognize EOD based on byte counts or based on special end of data characters.

Note *All of the ASCII-based filters accept an optional dictionary that includes the* ***CloseTarget*** *key for the encode filters and the* ***CloseSource*** *key for the decode filters.*

2.1 The ASCIIHexEncode and ASCIIHexDecode Filters

ASCIIHexEncode and ASCIIHexDecode filters are very simple filters whose most common purpose is in processing image data represented as hexadecimal quartets (two pairs of ASCII hexadecimal digits). Prior to LanguageLevel 2, the common technique for reading **image** or **imagemask** data was a procedure containing a **readhexstring** operator. The common method for processing an image in this manner used a PostScript language sequence similar to the following:

```
width height depth procedure image
```

where `procedure` would look something like this:

```
{currentfile tempstring readhexstring pop}
```

With the addition of **ASCIIHexDecode** (and image dictionaries), the sequence becomes much more simple to state, and more efficient. The image dictionary can now contain a **DataSource** key whose value can read as follows:

```
/DataSource currentfile /ASCIIHexDecode filter
```

Such a filter reads ASCII-encoded hexadecimal data, where each byte of input data specifies a hexadecimal quartet with a value between 0 and 15 (0–9 and either A–F or a–f).

The EOD symbol for **ASCIIHexEncode** and **ASCIIHexDecode** is the > character.

Adobe Systems Incorporated

Adobe Systems Incorporated

Example 1 creates an 300 by 300 dpi image whose pixels are random shades of gray. This simple example illustrates how an encoding filter is used to write data to an output file. The data is generated by a procedure, and the **writestring** operator is used to write the string data to the filter. The filter encodes the data and writes the encoded result to the final target.

Example 1 *Use of the ASCIIHexEncode Filter*

```
%!PS
% Define the output file name and a temp string
/data (hex.ps) (w) file def
/junk 1 string def
% stack is the output file
% 300 rows of 300 pixels
% Generate a random number, then use the lower 8 bits
% Write to the filter
/GenData {
    300 {
        300 {
            /s rand def
            junk 0 s 16#FF and put
             hex junk writestring
} repeat
} repeat
} def
% the output file is the target
% use ASCIIHexEncode as the filter
data
/ASCIIHexEncode filter
% define the file object
/hex exch def
% generate the data, which is written to the file
% object encoded by the filter
% and written to the output
GenData
% close the filter
hex closefile
```

Note *There are C language source code files available for the **ASCIIHexEncode** filter. These files are aschexec.c and protos.h. Although these files were written to accompany an earlier level of the PostScript language, their information and use should still apply. Source code can be acquired under license through the Adobe Developers Association.*

Note *For more information on the **ASCIIHexEncode** and **ASCIIHexDecode** filters, see Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition.*

2.2 The ASCII85Encode and ASCII85Decode Filters

ASCII85Encode and **ASCII85Decode** are also simple filters. They have the advantage of providing a small compression of the data (around 4:5) over and above a hexadecimal encoded file. Data in this format is encoded in base-85 format so that binary data can be encoded as seven-bit readable ASCII data. The main advantage of this format is that all the seven-bit ASCII characters can be safely transmitted over network connections that are not *eight-bit clean* (in other words, the data avoids the problem of interference by operating systems or communication channels that preempt use of control characters prevalent in eight-bit formats). ASCII85 is the encoding scheme used by the uuencode and uudecode programs available on UNIX systems. The EOD symbol for **ASCII85Encode** and **ASCII85Decode** is the ~> sequence of characters (tilde and "greater than" characters).

Example 2 shows the calling sequence to use an **ASCII85Decode** filter to decode a PostScript language stream.

Example 2 *Use of an ASCII85Decode Filter to Decode a PostScript Language Stream*

```
% A very small code sequence
...
currentfile /ASCII85Decode filter cvx exec
% The stream of data is here...
...
```

Note *There are C language source code files available for the **ASCII85Encode** and **ASCII85Decode** filters. These files are asc85dc.c, asc85ec.c, and protos.h. Although these files were written to accompany an earlier level of the PostScript language, their information and use should still apply. Source code can be acquired under license through the Adobe Developers Association.*

Note *For more information on the **ASCII85Encode** and **ASCII85Decode** filters, see Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition.*

Adobe Systems Incorporated

Adobe Systems Incorporated

3 LZW Filters

LZWEncode and **LZWDecode** are filters for encoding and decoding data according to the LZW (Lempel-Ziv-Walsh) compression scheme. LZW filters provide a lossless (no data loss) compression scheme. The syntax for using the LZWDecode filter is as follows:

```
target dict /LZWDecode filter
```

where *dict* is an optional dictionary.

Two new keys, in addition to **CloseSource**, have been added to the optional **LZWDecode** dictionary for LanguageLevel 3. Both of these keys refer to the form of the encoded data. They are shown in Table 4.

Table 4 *Keys in the LZWDecode Filter Dictionary*

Key	Type	
UnitSize	integer	optional
LowBitFirst	boolean	optional

UnitSize specifies the size of the units encoded by LZW. The only supported values are 2, 3, 4, 5, 6, 7, and 8. The default value for **UnitSize** is 8.

LowBitFirst specifies the *endianness* of the encoded byte stream. If the value is true, the encoded data is treated as *big-endian.* Big-endian means that the most significant bit or byte of data appears in the data stream before the next most significant bit or byte. If the value of **LowBitFirst** is false, the data is treated as little-endian. Little-endian means that the least significant bit or byte of data appears in the data stream before the next least significant bit or byte. The default value for the **LowBitFirst** key is false.

Note *The **LZWEncode** filter can take an optional dictionary with the **CloseTarget** key.*

The Flate filters (see Section 9.1), added in LanguageLevel 3, provide a similar method for data compression and are a patent-free alternative to LZW filters.

LZW compression has a worst-case expansion of at least a factor of 1.125, which can increase to a factor of nearly 1.5 in some implementations (plus the added effects of PNG tags, as with **FlateEncode** filters).

Note *For more information on the **LZWEncode** and **LZWDecode** filters, see Section 3.3 of the Supplement: PostScript Language Reference Manual, or Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition.*

4 RunLength Filters

RunLengthEncode and **RunLengthDecode** filters implement a very simple run-length encoding technique that provides reasonable compression at a low cost in performance, although its compression performance is not as good as either LZW or Flate. RunLength filters implement a *lossless* (no data loss) compression scheme that is similar to the Apple Macintosh PackBits routine. Data in a RunLength encoded file is represented as a sequence of *runs*. Each run consists of a length byte, followed by 1 to 128 data bytes. A length byte in the range of 0 to 127 indicates that the following *(length+1)* bytes are to be copied literally; that is, this run of bytes is not compressed. A length byte in the range 129 to 255 indicates that the following single byte is to be replicated *(257 – length)* times (that is, 2 to 128 times). This pair of bytes indicates a compressed run. A length byte whose value is 128 indicates the end of data (EOD).

Note *The **RunLength** filters now accept an optional dictionary.*

Example 3 shows how RunLength compression and ASCII base-85 encoding are applied to image data.

Example 3 *Use of the ASCII85Decode and RunLengthDecode Filters*

```
% A code sample
0 setgray
/rows 150 def
/cols 150 def
/bits 8 def
/mystream currentfile /ASCII85Decode filter
/RunLengthDecode filter def
/beginimage
{
    50 70 translate
    500 500 scale
    cols rows bits [cols 0 0 rows neg 0 rows]
    mystream image
} def
beginimage
% image data goes here...
```

Note *There are C language source code files available for RunLengthDecode filters. These files are runlendc.c, runlenec.c, and protos.h. Although these files were written to accompany an earlier level of the PostScript language, their information and use should still apply. Source code can be acquired under license through the Adobe Developers Association.*

Note *For more information on the **RunLengthEncode** and **RunLengthDecode** filters, see Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition.*

Adobe Systems Incorporated

Adobe Systems Incorporated

5 CCITTFax Filters

CCITT stands for the Comité Consultatif International Télégraphique et Téléphonique. This filter type, known as **CCITTFaxEncode** and **CCITTFaxDecode**, is used for encoding and decoding facsimile (FAX) data according to the CCITT standards. This filter is primarily intended for encoding and decoding image data, not for communicating with FAX machines.

Note *The **CCITTFax** filters accept an optional dictionary that includes the **CloseTarget** key for the encode filters and the **CloseSource** key for the decode filters.*

The implementation-defined limit for the value of the **Columns** key in the decode dictionary has been increased from 25,000 to 62,000 for LanguageLevel 3.

Note *There are C language source code files available for the **CCITTFaxEncode** and **CCITTFaxDecode** filters. These files are readme.fax, makefile, bitstm.h, ccmp.h, ccmpcode.h, ccmptab.h, cfaxfilt.h, protos.h, ccmpec.c, ccmpjc.c, flipbyte.c, revbits.c, and runtab0.c. Although these files were written to accompany an earlier level of the PostScript language, their information and use should still apply. Source code can be acquired under license through the Adobe Developers Association.*

Note *For more information on the **CCITTFaxEncode** and **CCITTFaxDecode** filters, see Section 3.3 of the Supplement: PostScript Language Reference Manual, or Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition. See also Technical Note #5128, "PostScript Level 2 and Fax Modem Printing."*

6 NullEncode Filter

NullEncode performs no data transformation; its output is identical to its input. Its primary function is to allow an arbitrary output target (file, procedure, or string) to be treated as an output file.

Note *The **NullEncode** filter accepts an optional dictionary that includes the **CloseTarget** key.*

Note *For more information on the **NullEncode** filter, see Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition.*

7 DCT Filters

DCT (Discrete Cosine Transform) filters are used for encoding and decoding grayscale or color image data in JPEG format. On LanguageLevel 3 printers that support web printing, the **DCTDecode** filter has been extended to decode progressive JPEG, a format which is widely used in web images. However, progressive JPEG is not part of LanguageLevel 3 and should not be used in LanguageLevel 3 programs; these programs should only use the Baseline Sequential JPEG variation. Decoding progressive JPEG images requires extra RAM to hold the complete image raster, typically 05, 1, or 2 times the size of the image in pixels, depending upon compression parameters. This memory requirement can be prohibitive for really large images.

Note *This encoding scheme is lossy and is not suitable for use with LanguageLevel 3 masked image data, or for image data in general.*

Example 4 shows a partial PostScript language program that compresses a left-to-right, top-to-bottom raster 3-color RGB image using the **DCTEncode** filter and writes the compressed image on another file.

Example 4 *Use of the DCTEncode Filter*

```
% Open a dictionary that contains optional parameters
jpeg begin
save mark 4 -2 roll
{
    /dest exch (w) file def % Open arg2 as output file
    /src exch (r) file def % Open arg1 as input file
    /Colors 3 def % Setup image-specific parameters
    /Columns 512 def
    /Rows 512 def
    /buf Columns Colors mul string def
    /filtdest dest jpeg /DCTEncode filter def
    Rows {
          filtdest src buf readstring pop writestring
        } repeat
    filtdest closefile dest closefile
} stopped {handleerror} if cleartomark restore
end % Close optional parameters dictionary
```

Adobe Systems Incorporated

Example 5 shows a partial PostScript language program that decodes and prints a JPEG-compressed file. The original photographic image was 24-bit RGB, 8 bits per component, 525 pixels high, 727 pixels wide, and 150 pixels per inch.

Example 5 *Decoding and Printing a JPEG-compressed File*

```
/DeviceRGB setcolorspace
126 270 translate % Center image on letter paper
349 252 scale % Scale image to original size
% Create a procedure to decode and image the
% DCT-encoded data. Note that 'exec' is followed by
% exactly one space character
{
    /Data currentfile /DCTDecode filter def
    <<
      /ImageType 1
      /Width 727
      /Height 525
      /ImageMatrix [727 0 0 -525 -525]
      /DataSource Data
      /BitsPerComponent 8
      /Decode [0 1 0 1 0 1]
    >> image
} exec
% Binary JPEG-encoded image data goes here...
showpage
```

Note *For more information on the **DCTEncode** and **DCTDecode** filters, see Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition. See also Technical Note #5116, "Supporting the DCT Filters in PostScript Level 2."*

Note *For more information on JPEG compression, see Technical Note #5083, "JPEG Technical Specification, Revision 9," and Technical Note #5095, "JPEG Source Vendor List."*

Adobe Systems Incorporated

8 SubFileDecode Filter

The **SubFileDecode** filter is an input source only filter. Its primary use is to break an arbitrary input stream into separate chunks. The ability to break the input file into chunks is based on being able to recognize specific end of data (EOD) strings. The syntax of the **SubFileDecode** filter has been changed in LanguageLevel 3 to the following:

```
source count string /SubFileDecode filter
```

This syntax has been extended to allow the following:

```
source << /EODCount count /EODString string>> /SubFileDecode
filter
```

If this filter will be used in combination with the **ReusableStreamDecode** filter, then the second form of syntax must be used.

Note *The **SubFileDecode** filter accepts an optional dictionary that includes the **CloseSource** key.*

Example 6 demonstrates how a **SubFileDecode** filter is used to read PostScript language code from the standard input file up to a specific marker, place the input into a file object, and execute the file object. The PostScript code to draw the first rectangle is defined in the subfile. The second rectangle is defined simply as PostScript code in the standard input file.

Example 6 *Use of the SubFileDecode Filter*

```
%!PS
currentfile 0 (%%EndOfExample) /SubFileDecode filter
/inch {72 mul} def
# Define a rectangle, set the color space and fill.
1 inch 6 inch moveto 6.5 inch 0 rlineto
0 4 inch rlineto -6.5 inch 0 rlineto closepath
/DeviceRGB setcolorspace 1.0 0.0 1.0 setcolor fill
% End of the subfile marker
%%EndOfExample
% Define the stream data and run the program
/data exch def
data cvx exec
% Create and show a separate rectangle
1 inch 1 inch moveto 6.5 inch 0 rlineto
0 4 inch rlineto -6.5 inch 0 rlineto closepath
/DeviceRGB setcolorspace 0.0 1.0 1.0 setcolor fill
showpage
```

Note *For more information on the **SubFileDecode** filter, see Section 3.3 of the Supplement: PostScript Language Reference Manual, or Sections 3.8.4 and 3.13 of the PostScript Language Reference Manual, Second Edition.*

Adobe Systems Incorporated

Adobe Systems Incorporated

9 New Filters for LanguageLevel 3

There are four new filters in LanguageLevel 3. They are the Flate filters, the **ReusableStreamDecode** filter, the **GIFDecode** filter, and the **PNGDecode** filter. In principle, reusable streams are a different kind of file object, but they are classified with filters. The GIF and PNG decode filters are considered optional filters and are only used on LanguageLevel 3 printers that support web printing.

9.1 FlateEncode and FlateDecode Filters

Flate filters are based on the DEFLATE compression scheme created by Jean-Loup Gailly and Mark Adler and described by Peter Deutsch in InterNic RFC 1950 (ZLIB Compressed Data Format Specification version 3.3) and InterNic RFC 1951 (DEFLATE Compressed Data Format Specification version 1.3). The DEFLATE compression format is based on a hybrid combination of the LZ77 (Lempel-Ziv 1977) algorithm and Huffman encoding; it specifies a lossless, compressed data format. The zlib specification named above describes a header which indicates the compression method. For the flate filter, this number is eight, which indicates DEFLATE.

The **FlateDecode** filter decodes data that is encoded with the zlib/DEFLATE compression scheme (as is done with ZIP files). This format has an end of data (EOD) marker. The **FlateDecode** filter is used with the **filter** operator as follows:

```
source dictionary /FlateDecode filter
```

The **FlateEncode** filter is used with the filter operator as follows:

```
target dictionary /FlateEncode filter
```

The **FlateEncode** filter encodes binary or ASCII data, optionally after pre-transformation by a predictor function, and always produces binary data.

Table 5 shows the keys used in the optional dictionary for a **FlateEncode** filter.

Table 5 *Keys in the FlateEncode Filter Dictionary*

Key	Type	
CloseTarget	Boolean	optional
Effort	integer	optional
Predictor	integer	optional
Columns	integer	optional
Colors	integer	optional
BitsPerComponent	integer	optional

If the value of **CloseTarget** is true, the output stream is closed when the data target is closed. The default value of this key is false.

Effort controls the memory used for Flate compression and the execution speed of the compression. Supported values for this key are −1 to 9, inclusive. A value of 0 compresses rapidly, but not tightly using little auxiliary memory (this means that fast execution is possible, but the compression factor will not be as good as with a higher value. A higher value means better compression, but slower execution and greater RAM usage.). A value of 9 compresses slowly but as tightly as possible, using as much auxiliary memory as is necessary. The default value for this key is −1, which means to map it to a value in the range [0,9] that is a reasonable default for this implementation (that is, a value that is based on available RAM).

Columns specifies the number of samples in a sampled row. The value of this key only has an effect on the filter if the value of **Predictor** is greater than 1. See the description of the **Predictor** key, below. The default value for **Columns** is 1.

The **Colors** key specifies the number of interleaved color components in a sample. The default value of this key is 1. Again, this key only has an effect on the filter if the value of **Predictor** is greater than 1.

BitsPerComponent specifies the number of bits used to represent each color component. The only supported values for this key are 1, 2, 4, 8, and 16. The default value is 8.

Note *Image data defined with 16 bits per component may not be used directly as input to the* **image** *operator.*

Adobe Systems Incorporated

Table 6 shows the keys used in the dictionary for a **FlateDecode** filter.

The **Predictor** key is discussed below.

Table 6 *Keys in the FlateDecode Filter Dictionary*

Key	Type	
CloseSource	Boolean	optional
Predictor	integer	optional
Columns	integer	optional
Colors	integer	optional
BitsPerComponent	integer	optional

If the value of **CloseSource** is true, the output stream is closed when the data source is closed. The default value for this key is false.

The **Columns, Colors**, and **BitsPerComponent** keys are defined the same as for the Flate encode filter.

Predictor Functions

Like the LZW encode filter, the Flate encode filter compresses more compactly if its input data is highly predictable. One way of increasing the predictability of many continuous-tone sampled images is to replace each pixel with the difference between that pixel and some predictor function applied to earlier neighboring pixels. If the predictor function works well, the post-prediction data will cluster toward zero.

Two predictor function groups are supported. The first is the TIFF group, which consists of the single function that is Predictor 2 in the TIFF standard. TIFF Predictor 2 predicts that each color component of a pixel will be the same as the corresponding color component of the pixel immediately to the left.

The **Predictor** key in the Flate filter dictionary selects the predictor function.To select the TIFF Predictor 2 predictor function, the value of the **Predictor** key should be 2.

Adobe Systems Incorporated

The second group of predictor functions is the PNG group, which consists of the filters or predictors of the World Wide Web Consortium (W3C) Portable Network Graphics (PNG) recommendation. There are five basic PNG predictor algorithms, and a sixth one that is a hybrid of the first five. These can be set in the Flate filter dictionary as follows:

- A **Predictor** key value of 10 uses no PNG prediction. That is the value is set to *None*.
- A **Predictor** key value of 11 uses *Sub* prediction, which means to predict the same value as the pixel to the left.
- A **Predictor** key value of 12 uses *Up* prediction, which means to predict the same value as the pixel above.
- A **Predictor** key value of 13 uses *Average* prediction, which means to predict the average of the value of the pixel above and the pixel to the left.
- A **Predictor** key value of 14 uses *Paeth* prediction, which means to predict a value that is a non-linear function of the pixel above, the pixel to the left, and the pixel to the upper left.
- A **Predictor** key value of 15 uses the *Optimum* prediction, which is the hybrid of the first five algorithms.

The default value for the **Predictor** key is 1, which means that no prediction at all is made.

The TIFF and PNG predictor groups have some similarities. Both assume that image data is presented in order, from top row to bottom row, and from left to right within a row. Both assume that a row occupies a whole number of bytes, rounded upward as necessary. Both assume that pixels and their color components are packed into bytes from high- to low-order bits (big-endian). Both assume that all color components of pixels outside the image are 0; these pixels are necessary for predictions near the boundaries of images.

The two predictor groups also have several significant differences. The post-prediction data for each PNG-predicted row begins with an explicit algorithm tag, so different rows can be predicted with different algorithms to improve compression. TIFF 2 prediction has no tags, so the same algorithm applies to all rows of data. The TIFF function group predicts each color component from the prior instance of that color component without regard to the width of the color component or the number of colors. The PNG function group predicts each byte from the corresponding byte of the prior pixel (Sub algorithm) and/or the same pixel on the prior line (Up algorithm) and/or the prior pixel on the prior line (Average algorithm). This happens regardless of

Adobe Systems Incorporated

whether there are multiple color components in a byte, or whether a single color component spans multiple bytes. This approach can result in significantly better compression speed but with somewhat less compression.

Comparison of LZW and Flate Encoding

Flate encoding, like LZW encoding, discovers and exploits many patterns in its input data, whether it be text or images. Because of its cascaded adaptive Huffman coding, Flate-encoded output is usually substantially more compact than LZW-encoded output for the same input.

Flate and LZW decoding speeds are comparable, but the Flate *encoding* speed is considerably slower than LZW encoding. In most cases, the Flate and LZW encode filters compress their inputs substantially. In the worst case, the **FlateEncode** filter expands its input by no more than a factor of 1.003, plus the added effects of algorithm tags added by PNG predictors and the added effects of any explicit **flushfile** operations. LZW compression has a worst-case expansion of at least a factor of 1.125, which can increase to a factor of nearly 1.5 in some implementations (plus the added effects of PNG tags).

Note *For more information on the **FlateEncode** and **FlateDecode** filters, see Sections 3.3 of the Supplement: PostScript Language Reference Manual.*

9.2 Reusable Streams and the ReusableStreamDecode Filter

Most PostScript language streams are consumed in a serial order, and once consumed, they cannot be read again. In LanguageLevel 2, in order to randomly or repeatedly access a stream of data, the data must be read into a string or written to a file on a storage device such as a hard disk. Although a string is repositionable to accommodate these actions, its size is limited to 64Kb (in previous levels of the PostScript language, it was possible to implement an “in-memory” file out of an array of strings, although this method was never recommended). In LanguageLevel 3, a new type of stream, called a *reusable stream*, has been introduced to work around the limitations of using streams or strings. Reusable streams do not impose a size limitation on storage; the amount of data that can be stored is limited only by the amount of storage (VM) available on the printer/device.

Reusable streams provide a new kind of file object that can be positioned to arbitrary points in the data stream, and whose contents can be read more than once. This functionality can be used to handle such cases as image data that is replicated multiple times on a page or image data in forms, and function data or mesh data used in **Shading** dictionaries for **PatternType 2** patterns.

Note *See Technical Note #5600, “Smooth Shading,” for more information on **Shading** dictionaries and **PatternType 2** pattern dictionaries.*

Adobe Systems Incorporated

Reusable streams differ from other types of file objects in the following ways:

- When an EOD or EOF is encountered in a reusable stream, the file is not closed; the file must be closed explicitly.
- Reusable streams automatically use a special device, %ram%, as the underlying file in which the data is stored for random access.

Note *See Section 10.5 of the Supplement: PostScript Language Reference Manual, for more information on ramdisks and the %ram% device.*

A reusable stream can be created using the **filter** operator and the new decode filter called **ReusableStreamDecode**. The **ReusableStreamDecode** filter can be used in one of two ways. Both methods return a file object on the operand stack.

One way, and probably the simplest, is to specify just a data source. The syntax of this method is as follows:

```
source /ReusableStreamDecode filter
```

An example use this method would be to read image data from **currentfile** and then create a file object that can be randomly accessed and repositioned. The code for this would look like the following:

```
currentfile /ASCIIHexDecode filter
/ReusableStreamDecode filter
...image data follows the filter operation...
/data exch def
```

In this particular example, the **ASCIIHexDecode** filter senses its EOD because of the > marker at the end of the data. In cases where the end data is not so readily identifiable, the input file should be filtered through a **SubFileDecode** filter prior to the reusable stream filter.

The second method involves an optional dictionary argument in addition to the data source. The syntax is as follows:

```
source dictionary ReusableStreamDecode filter
```

The example use of ReusableStreamDecode above could be rewritten for this second method as follows:

```
currentfile << /Filter /ASCIIHexDecode >>
/ReusableStreamDecode filter
```

Adobe Systems Incorporated

Adobe Systems Incorporated

The use of the **filter** operator has some unusual side effects, and the resulting `fileobj` has some unusual attributes. When **filter** is executed, the data from `source` may or may not be immediately buffered in virtual memory (VM) or written to disk, depending on a variety of factors, including the following:

- The nature of *source* – whether it is **currentfile**, a disk file, a string, a procedure, or a filtered file.
- The availability of system disk storage.
- The availability of VM, constrained by the ramdisk **LogicalSize** parameter.
- The set of **Filters** specified in `dictionary`.
- Implementation and system memory management details.

If *source* is derived from **currentfile**, or from a PostScript procedure, the data will always be read at the time the **filter** operator is executed. **currentfile** data should typically be filtered through a **SubFileDecode** filter. If *source* is a string, or if *source* is derived from a disk-based file, that string or file should be treated as read-only; writing into this string or file will have unpredictable consequences for the data read from *fileobj*. However, such strings may be undefined, although they will not be garbage collected until they are no longer needed. Also, in many file systems, such files may be deleted, although their disk space will be freed only when it is no longer needed.

When the reusable stream filter has read all of the data, it leaves a file object on the operand stack. Unlike other filtered files, this file object can be re-positioned just like a random-access file. Bytes of the file are indexed from 0 up to *(length -1)*. The file object is not closed automatically.

The file object has a *length* value. The length of the file object can be obtained with the following code:

```
data flushfile % set file to EOF
data fileposition
```

Here are the file operations that can be performed on the file object that is returned as a result of a **ReusableStreamDecode** filter:

- **closefile**: closes the file object. The file object is also closed when it is destroyed by the **restore** operator or garbage collection. Any associated temporary file created on a file system will be deleted when the file object is closed.

- **bytesavailable**: returns the file object size minus the current file position. If the file is currently positioned at EOF, 0 is returned.

- **flushfile**: sets the file position to EOF.

- **resetfile**: resets the file position to 0. This is a convenience operator for the more lengthy, but explicit call:

  ```
  0 setfileposition
  ```

- **setfileposition**: sets the file object position to any value between 0 and *length*. Note that setting the file position to *length* effectively sets it to EOF. A file position less than 0 or greater than length returns an error.

- **fileposition**: returns the current position of the file object. The result is always in the range 0 through length, where a position of length means the file is at EOF.

Table 7 lists the keys available in the optional **ReusableStreamDecode** dictionary. Explanations of each key are given below.

Table 7 *Keys in the ReusableStreamDecode Dictionary*

Key	Type	
Filter	name or array	optional
DecodeParams	array of variable	optional
Intent	integer	optional
AsyncRead	boolean	optional
CloseSource	boolean	optional

Filter specifies any filter that is to be applied before delivering data to the reader. The value of the key can be either the name of a single decode filter, or it can be an array of decode filter names. Multiple filters are applied to the incoming data in the order in which they are specified in the array. For example, data compressed using the **LZWEncode** and then the **ASCII85Encode** filters are then decoded by providing the following array:

```
[/ASCII85Decode /LZWDecode]
```

Adobe Systems Incorporated

DecodeParams specifies the parameters associated with each of the filters listed in the **Filter** array. If **Filter** contains no elements or is missing, then **DecodeParams** is not needed. If **Filter** contains one element (that is, its type is name), then **DecodeParams** will either contain one dictionary or a null object. If the **Filter** array contains more than one element, **DecodeParams** is an array. Each element (parameter) of the array must have a one-to-one correspondence with the elements of the **Filter** array. The value of each element (parameter) is either a dictionary object or a null object.

Note *The **DCTDecode** and **SubFileDecode** filters require a dictionary object.*

The value of the **Intent** key specifies a hint at the intended purpose of the reusable data. If the value is recognized, it may help optimize the data storage or caching strategy used. The currently supported values for the **Intent** key are as follows:

- 0: specifies image data (the default value)
- 1: specifies image mask data
- 2: specifies sequentially-accessed table look-up data, such as threshold arrays
- 3: specifies randomly-accessed table look-up data, such as functions, CID fonts, and color rendering dictionaries

Note *If the value of **Intent** is not recognized, it is ignored.*

If the value of the **AsyncRead** key is false, the file position of the input stream (source) is advanced to end of file (EOF) or end of data (EOD). This key only affects disk files.

If the value of the **CloseSource** key is true, the input stream (source) is closed when the reusable stream is closed.

Adobe Systems Incorporated

Example 7 shows a typical use of a reusable stream for a masked image.

Example 7 *Use of the ReusableStreamDecode Filter*

```
%!PS-Adobe-3.0
% From MASKIM33.PS
% This example illustrates Type 3 Masked Images with
% Type 3 interleave. In order to use the image data
% and mask data twice, reusable streams or needed.
% Note that the common practice is to actually have
% only the mask data in a reusable stream.
currentfile /ASCIIHexDecode filter /ReusableStreamDecode
filter
...% Mask data goes here
>
/maskstream exch def
currentfile /ASCIIHexDecode filter /ReusableStreamDecode
filter
% Image data goes here
>
/datastream exch def
/inch {72 mul} def
/DeviceRGB setcolorspace
% Create the image data dictionary
/ImageDataDictionary 8 dict def
ImageDataDictionary begin
    /ImageType 1 def
    /Width 317 def
    /Height 299 def
    /BitsPerComponent 8 def
    /DataSource datastream def
    /MultipleDataSources false def
    /ImageMatrix [317 0 0 299 0 0] def
    /Decode [0 1 0 1 0 1] def
end
% Create the mask data dictionary
/ImageMaskDictionary 8 dict def
ImageMaskDictionary begin
    /ImageType 1 def
    /Width 317 def
    /Height 299 def
    /BitsPerComponent 1 def
    /DataSource maskstream def
    /MultipleDataSources false def
    /ImageMatrix [317 0 0 299 0 0] def
    /Decode [0 1] def
end
% code continued on the next page
```

Adobe Systems Incorporated

Adobe Systems Incorporated

```
% Now create the masked image dictionary
/MaskedImageDictionary 7 dict def
MaskedImageDictionary begin
    /ImageType 3 def
    /InterleaveType 3 def
    /MaskDict ImageMaskDictionary def
    /DataDict ImageDataDictionary def
end
% Draw the masked image in the first location
gsave
    2.05 inch 5.5 inch translate
    4.4 inch 4.15 inch scale
    % Rewind the reusable mask and data streams
    maskstream resetfile
    datastream resetfile
    MaskedImageDictionary image
grestore
% Draw the masked image in the second location
gsave
    2.05 inch 0.5 inch translate
    4.4 inch 4.15 inch scale
    maskstream resetfile
    datastream resetfile
    ImageMaskDictionary /Decode [1 0] put
    MaskedImageDictionary image
grestore
showpage
```

Note *For more information on the **ReusableStreamDecode** filter, see Sections 3.3 of the Supplement: PostScript Language Reference Manual.*

9.3 GIFDecode Filter

The **GIFDecode** filter decodes image data that is stored in the GIF (Graphics Interchange Format). The filter deciphers all the header information associated with that format and delivers the image data, either in RGB or indexed format. The main function of the **GIFDecode** filter is to assist in the HTML to PostScript translation available in *web-ready* LanguageLevel 3 printers (a web-ready printer is one that can readily handle the display or printing of web-based material).

GIF supports a number of features that make using this filter for inlined images in a PostScript language document undesirable:

- GIF supports interlaced images. The problem with interlaced images is that the scan lines do not appear in either top-down or bottom-up order; the scan lines are out of order. The **GIFDecode** filter, however, delivers the scan lines in the order in which they are encountered in the GIF file, and, for interlaced images, this filter is unsuitable as an input source to the **image** operator.

- GIF files can contain multiple images (usually for animation purposes). The **GIFDecode** filter delivers data only for the first image in a multi-image file. Attempts to read beyond the end of the first image result in an EOF condition on the filter stream.

- The **GIFDecode** filter does not do exactly what its name implies. GIF is not just a data encoding representation like LZW; it is also an image data format. In order to make use of a GIF image, the user must extract the image parameters from the file (the methods of which are not provided with the filter). The user must also deal with out-of-order data for interlaced GIF images.

Note *The **GIFDecode** filter accepts an optional dictionary that includes the **CloseSource** key.*

Note *The **GIFDecode** filter is considered an optional filter used only as part of the implementation of web printing. PostScript language programs should not invoke this decode filter.*

Adobe Systems Incorporated

Adobe Systems Incorporated

9.4 PNGDecode Filter

The **PNGDecode** filter has been created for use on web-ready LanguageLevel 3 printers. The purpose of this filter is to aid in the printing of PNG images. PNG images may need to be printed because they are referenced in an HTML file, or because the user submitted a URL that directly references a PNG image.

Similar to **GIFDecode**, the **PNGDecode** filter does not do exactly what the name implies. The user must be able to handle image parameters, out-of-order data, and data buffering, as explained above for **GIFDecode**.

Note *The **PNGDecode** filter accept an optional dictionary that includes the **CloseSource** key.*

Note *The **PNGDecode** filter is considered an optional filter used only as part of the implementation of web printing. PostScript language programs should not invoke this decode filter.*

Appendix A

Bibliography of Outside Sources

While this in not an exhaustive list of references for filters, it will give the reader some sources for types of filters covered in this document.

For a complete specification on DEFLATE and zlib, see the documentation section of the official zlib web site at http://quest.jpl.nasa.gov/Zlib.

For complete documenation on the PNG format, see the PNG web site at http://quest.jpl.nasa.gov/PNG.

For information on the DEFLATE Compressed Data Format, see the specification by Peter Deutsch on the web site at http://www.internic.net/rfc/rfc1951.txt.

Graphics International Format, Version 89a, © 1990 by CompuServe Incorporated, Columbus, Ohio, provides information on GIF images.

Adobe Systems Incorporated

Adobe Systems Incorporated

Index

A

ASCII85Decode 17
ASCII85Encode 17, 31
ASCIIHexDecode 15, 29
ASCIIHexEncode 15
AsyncRead 32

B

Baseline Sequential JPEG 21
Big-Endian 18, 27
BitsPerComponent 25, 26
bytesavailable 31

C

CCITT 20
CCITTFaxDecode 20
CCITTFaxEncode 20
closefile 13, 31
CloseSource 13, 15, 18, 20, 23, 26, 32, 35, 36
CloseTarget 13, 15, 18, 20, 25, 35
Colors 25, 26
Columns 20, 25, 26
currentfile 11, 29, 30

D

DataSource 15
DCT 21
DCTDecode 21
DCTEncode 21
DecodeParams 32
DEFLATE 14, 24

E

Effort 25
Endianness 18

F

FAX ix, 20
fileposition 31
Filter 12, 13, 31, 32
filter 24, 29, 30
Filters 30
Flate 14, 18, 19, 24
FlateDecode x, 14, 24, 26
FlateEncode x, 18, 24, 28
flushfile 28, 31

G

GIF 14, 35
GIFDecode x, 12, 14, 24, 35, 36

H

HTML 35, 36

I

image 15, 25, 35
imagemask 15
Intent 32

J

JPEG ix, 21

L

LanguageLevel 2 13, 15, 28
LanguageLevel 3 ix, x, 11, 12, 13, 18, 20, 21, 23, 24, 28
Little-Endian 18
LogicalSize 30

Adobe Systems Incorporated

LowBitFirst 18
LZW 14, 18, 19, 28, 35
LZWDecode ix, 18
LZWEncode ix, 18, 31

N

NullEncode 12

P

PatternType 2 28
PNG 14, 18, 27, 36
PNGDecode x, 12, 14, 24, 36
Predictor 25, 26, 27
Predictor Function 26
Progressive JPEG 21

R

readhexstring 15
resetfile 31
resourceforall 12
resourcestatus 12
restore 13, 31
ReusableStreamDecode x, 23, 24, 29, 31
RunLengthDecode ix, 19
RunLengthEncode ix, 19

S

setfileposition 31
Shading 28
SubFileDecode 23, 29, 30

T

TIFF 14, 26, 27

U

UnitSize 18
UNIX 17
URL 36
uudecode 17
uuencode 17

W

writestring 16

Z

ZIP 24
zlib 14, 24

Adobe Systems Incorporated