

**Usage**  
Storage of digital motion picture data in a bitmap file format.

**Comments**  
DPX is a modification of the Kodak Cineon format. The Society of Motion Picture and Television Engineers (SMPTE) added header information to DPX not present in the original Kodak Cineon format.

DPX is a bitmap file format used to store a single frame of a motion picture or video data stream. Multiple DPX files are used to store and exchange digital moving picture sequences between a wide variety of electronic and computer systems.

**Contents:**  
[File Organization](https://www.fileformat.info/format/dpx/egff.htm" \l "DPX-DMYID.2)  
[File Details](https://www.fileformat.info/format/dpx/egff.htm" \l "DPX-DMYID.3)  
[For Further Information](https://www.fileformat.info/format/dpx/egff.htm" \l "DPX-DMYID.4)

The DPX format is an ANSI and SMPTE standard based on the Kodak Cineon file format. These two formats are nearly identical, except for several extra header fields defined in the DPX format. For this reason, we recommend that you support the DPX format even if you are just interested in parsing Cineon image files.

DPX has several features designed to support device and resolution independence. For example, DPX files, like TIFF files, may be written using either the big- or little-endian byte-ordering schemes. Image data is stored as a series of up to eight elements using one of the many color component formats supported by the DPX standard. Pixel data itself may be stored a depth of 1, 8, 10, 12, 16, 32, or 64 bits. Padding of scan lines and image data, and the packing of pixel data, is also supported.

## File Organization

DPX files are organized as a series of headers followed by an optional block of user-defined data and finally by the image data itself. The DPX standard defines two headers:

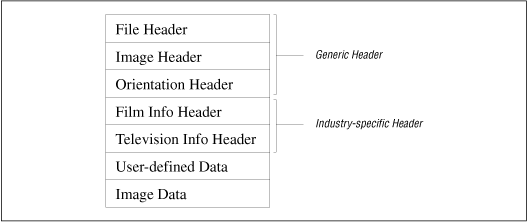
* The generic header containing information on the DPX file, the image data, and the image orientation
* The industry-specific header, containing information used by the film and television industries

For clarity, this article will regard these sections as five separate headers.

Following these headers is an optional area containing user-defined data. This area may contain any type of information the DPX file writer needs to store this data and may be up to one megabyte in size.

Last in the file is the image data itself, stored as an array of pixel component values. Figure DPX-1 illustrates the internal arrangement of a DPX file.

#### Figure DPX-1: DPX format



## File Details

The five DPX headers are all fixed in size and contain fields ranging in length from one to several hundred bytes. Fields of the SINGLE (16-bit) and DOUBLE (32-bit) floating-point data types are also common.

Any integer or real header fields that contain undefined data are to be initialized to their "all ones" value. In C, this is most easily accomplished by assigning the one's-complement of 0 to an undefined field, as follows:

WORD val = ~0; /\* Set val to 0xFFFF \*/

Undefined ASCII fields are initialized to all NULL (ASCII 00h) characters.

### File Header

The file header contains information on the DPX file and all of its parts, including the name, size, and date stamp of the file, and the internal location of the image data. The DPX file header is 768 bytes in length and has the following format:

typedef struct \_GenericFileHeader

{

DWORD Magic; /\* Magic number \*/

DWORD ImageOffset; /\* Offset to start of image data in bytes \*/

char Version[8]; /\* Version stamp of header format \*/

DWORD FileSize; /\* Total DPX file size in bytes \*/

DWORD DittoKey; /\* Image content specifier \*/

DWORD GenericSize; /\* Generic section header length in bytes \*/

DWORD IndustrySize; /\* Industry-specific header length in bytes \*/

DWORD UserSize; /\* User-defined data length in bytes \*/

char FileName[100]; /\* Name of DPX file \*/

char TimeDate[24]; /\* Time and date of file creation \*/

char Creator[100]; /\* Name of file creator \*/

char Project[200]; /\* Name of project \*/

char Copyright[200]; /\* File contents copyright information \*/

DWORD EncryptKey; /\* Encryption key \*/

char Reserved[104]; /\* Reserved field used for padding \*/

} GENERICFILEHEADER;

Magic is the DPX file identification value. It is also the indicator of the byte order of the DPX file data. If this value is 53445058h ("SDPX"), the file data is written in big-endian order. If this value is 58504453h ("XPDS"), the file data is written in little-endian order.

ImageOffset is the offset of the start of the image data from the beginning of the file in bytes.

Version is an 8-byte ASCII field containing a string of the version of the DPX standard to which the file conforms. For v1.0 files, this field contains the ASCII string "V1.0".

FileSize is the total size of the DPX file in bytes.

DittoKey indicates whether the image in the DPX file is the same as in the previous frame (a value of 0), or is a new frame (a value of 1).

GenericSize is the total size of the generic header section in bytes.

IndustrySize is the total size of the industry-specific header section in bytes.

UserSize is the total size of the user-defined data area in bytes. If no user-defined area is present, its value will be 0.

FileName is a 100-byte ASCII field containing free-form text information identifying the name and path of the DPX file.

TimeDate is a 24-byte field containing a time and date stamp of when the DPX file was created. This stamp is a string in the form *YYYY:MM:DD:HH:MM:SS:LTZ*.

Creator is a 100-byte ASCII field containing free-form text information identifying the creator of the DPX file and/or the image it contains.

Project is a 200-byte ASCII string containing free-form text information identifying the project with which the DPX file and/or image is associated.

Copyright is a 200-byte ASCII string containing free-form text information identifying the copyright of the image data stored within the DPX file.

EncryptKey contains a 4-byte value used as a key to process the file using a user-specified decryption algorithm. EncryptKey has a value of FFFFFFFFh if the image data is not encrypted. The DPX header data is never encrypted.

Reserved is a 104-byte field used to pad the header out to 768 bytes in length. Future versions of the DPX format may support additional fields in this area.

### Image Header

Immediately following the file header is the image header. This header stores information specific to the image data of the frame stored in the DPX file. Note that this header contains a sequence of eight identical subheaders. Each subheader stores information on each of the eight possible elements that make up the image data.

The image header is 640 bytes in size and has the following format:

typedef struct \_GenericImageHeader

{

WORD Orientation; /\* Image orientation \*/

WORD NumberElements; /\* Number of image elements \*/

DWORD PixelsPerLine; /\* Pixels per line \*/

DWORD LinesPerElement; /\* Lines per image element \*/

struct \_ImageElement

{

DWORD DataSign; /\* Data sign extension \*/

DWORD LowData; /\* Reference low data code value \*/

SINGLE LowQuantity; /\* Reference low quantity represented \*/

DWORD HighData; /\* Reference high data code value \*/

SINGLE HighQuantity; /\* Reference high quantity represented \*/

BYTE Descriptor; /\* Descriptor for image element \*/

BYTE Transfer; /\* Transfer characteristics for element \*/

BYTE Colorimetric; /\* Colormetric specification for element \*/

BYTE BitSize; /\* Bit size for element \*/

WORD Packing; /\* Packing for element \*/

WORD Encoding; /\* Encoding for element \*/

DWORD DataOffset; /\* Offset to data of element \*/

DWORD EndOfLinePadding; /\* End of line padding used in element \*/

DWORD EndOfImagePadding; /\* End of image padding used in element \*/

char Description[32]; /\* Description of element \*/

} ImageElement[8];

BYTE Reserved[52]; /\* Reserved field used for padding \*/

} GENERICIMAGEHEADER;

Orientation specifies the proper direction of the image on the display. Line direction and frame direction are specified, respectively, by the values:

|  |  |
| --- | --- |
| 0 | Left to right, top to bottom |
| 1 | Right to left, top to bottom |
| 2 | Left to right, bottom to top |
| 3 | Right to left, bottom to top |
| 4 | Top to bottom, left to right |
| 5 | Top to bottom, right to left |
| 6 | Bottom to top, left to right |
| 7 | Bottom to top, right to left |

0 is the only value supported for the core set file format.

NumberElements indicates the number of elements in the image data. This value also indicates the number of elements in the ImageElement[] array that contain valid data.

PixelsPerLine specifies the number of pixels in a scan line. There is an equal number of pixels in each line per element.

LinesPerElement specifies the number of lines in the element. There is an equal number of lines in each element.

The following 15 fields are repeated eight times in the generic image header. Each of these 72-byte subheaders contain information on a single element of the image data. The fields of all eight subheaders will always be present in each DPX file, even if there is only one element of image data.

DataSign is 0 if the image data is unsigned and 1 if the data is signed. Image data is normally stored using unsigned values.

LowData defines the minimum expected color value.

LowQuantity defines the reference for the low quantity value specified in the LowData field. This reference value should be considered the lowest possible value and is typically 0.

HighData defines the maximum expected color value.

HighQuantity defines the reference for the high-quantity value specified in the HighData field. This reference value should be considered the highest possible value, and is typically the largest value that can be stored in a component of the specified size (for example, 256 for 8-bit components).

Descriptor specifies the type of component stored by the element and its pixel-packing order. There are 256 possible values to this field; the following are defined:

|  |  |
| --- | --- |
| 0 | User-defined |
| 1 | Red |
| 2 | Green |
| 3 | Blue |
| 4 | Alpha |
| 6 | Luminance |
| 7 | Chrominance |
| 8 | Depth |
| 9 | Composite video |
| 50 | RGB |
| 51 | RGBA |
| 52 | ABGR |
| 100 | CbYCrY |
| 101 | CbYaCrYa |
| 102 | CbYCr |
| 103 | CbYCra |
| 150 | User-defined 2-component element |
| 151 | User-defined 3-component element |
| 152 | User-defined 4-component element |
| 153 | User-defined 5-component element |
| 154 | User-defined 6-component element |
| 155 | User-defined 7-component element |
| 156 | User-defined 8-component element |

All values not defined are reserved for use in future versions of DPX.

Transfer describes the transfer characteristics used to transform the data from a linear original. The possible values of this field are:

|  |  |
| --- | --- |
| 0 | User-defined |
| 1 | Printing density |
| 2 | Linear |
| 3 | Logarithmic |
| 4 | Unspecified video |
| 5 | SMPTE 240M |
| 6 | CCIR 709-1 |
| 7 | CCIR 601-2 system B or G |
| 8 | CCIR 601-2 system M |
| 9 | NTSC composite video |
| 10 | PAL composite video |
| 11 | Z linear |
| 12 | Z homogeneous |

Colorimetric defines the color reference primaries for color additive systems, or color responses for color subtractive systems. The possible values of this field are:

|  |  |
| --- | --- |
| 0 | User-defined |
| 1 | Printing density |
| 4 | Unspecified video |
| 5 | SMPTE 240M |
| 6 | CCIR 709-1 |
| 7 | CCIR 601-2 system B or G |
| 8 | CCIR 601-2 system M |
| 9 | NTSC composite video |
| 10 | PAL composite video |

BitSize specifies the number of bits in each component of the image element. Valid values are 1, 8, 10, 12, 16, 32, and 64.

Packing defines the method used to pack component data in each line of the image element data. Possible values are 0 (packed 32-bit words) and 1 (filled 32-bit words).

Encoding indicates whether the image element data is run-length encoded or not. Values are 0 (not encoded) and 1 (run-length encoded).

DataOffset indicates the offset to the start of the image element data in bytes from the beginning of the file.

EndOfLinePadding specifies the number of bytes added as padding to the end of each scan line. It is common to pad each line out to the nearest 32-bit boundary. The default value is 0, indicating that no end-of-line padding is present.

EndOfImagePadding specifies the number of bytes added as padding to the end of the image data. It is common to pad the element data out to an even 8K block boundary. The default value is 0, indicating that no end-of-image padding is present.

Description is a 32-byte field storing an ASCII string containing free-form text information describing the contents of the element.

Reserved is a 52-byte field used to pad the header out to 640 bytes in length. Future versions of the DPX format may support additional fields in this area.

### Orientation Header

The last of the generic headers is the orientation header. Information in this header describes the position of the image on the display and also contains some additional information on the source of the image data.

This header is 256 bytes in length and has the following format:

typedef struct \_GenericOrientationHeader

{

DWORD XOffset; /\* X offset \*/

DWORD YOffset; /\* Y offset \*/

SINGLE XCenter; /\* X center \*/

SINGLE YCenter; /\* Y center \*/

DWORD XOriginalSize; /\* X original size \*/

DWORD YOriginalSize; /\* Y original size \*/

char FileName[100]; /\* Source image file name \*/

char TimeDate[24]; /\* Source image date and time \*/

char InputName[32]; /\* Input device name \*/

char InputSN[32]; /\* Input device serial number \*/

WORD Border[4]; /\* Border validity (XL, XR, YT, YB) \*/

DWORD AspectRatio[2]; /\* Pixel aspect ratio (H:V) \*/

BYTE Reserved[28]; /\* Reserved field used for padding \*/

} GENERICORIENTATIONHEADER;

XOffset and YOffset indicate the offset of the first pixel in the stored image from the first pixel in the original image. If the two images are not offset, then the values of these fields will be 0.

XCenter and YCenter are floating-point values indicating the X and Y coordinates of the center of the image.

XOriginalSize and YOriginalSize indicate the width and height, respectively, of the original image, in pixels per line and lines per image.

FileName is the name of the source image from which the image was processed or extracted.

TimeDate is a 24-byte field containing a time and date stamp of when the source image was created. This stamp is a string in the form *YYYY:MM:DD:HH:MM:SS:LTZ*.

InputName is a 32-byte ASCII string declaring the name of the input device.

InputSN is a 32-byte ASCII string declaring the serial number of the input device.

Border is a set of four values that describe the region of the image eroded due to edge-sensitive filter processing. These values are, in order: X left, X right, Y top, and Y bottom. Values of 0, 0, 0, 0 indicate no border and therefore no erosion.

AspectRatio is the pixel aspect ratio described as a horizontal value (AspectRatio[0]) divided by a vertical value (AspectRatio[1]).

Reserved is a 28-byte field used to pad the header out to 256 bytes in length. Future versions of the DPX format may support additional fields in this area.

### Industry-Specific Headers

Following the generic headers are two headers that contain industry-specific information. Both of these headers are always present in every DPX file and are initialized with undefined field values when not used. These headers do not contain any information required by the DPX standard and their use is optional.

#### Film Information Header

The first industry header contains information used by the motion picture film industry. The information in this header describes the film and camera source from which the image frame data was derived. This header is 256 bytes in size and has the following format:

typedef struct \_IndustryFilmInfoHeader

{

char FilmMfgId[2]; /\* Film manufacturer ID code \*/

char FilmType[2]; /\* File type \*/

char Offset[2]; /\* Offset in perfs \*/

char Prefix[6]; /\* Prefix \*/

char Count[4]; /\* Count \*/

char Format[32]; /\* Format \*/

DWORD FramePosition; /\* Frame position in sequence \*/

DWORD SequenceLen; /\* Sequence length in frames \*/

DWORD HeldCount; /\* Held count \*/

SINGLE FrameRate; /\* Frame rate of original in frames/sec \*/

SINGLE ShutterAngle; /\* Shutter angle of camera in degrees \*/

char FrameId[32]; /\* Frame identification \*/

char SlateInfo[100]; /\* Slate information \*/

BYTE Reserved[56]; /\* Reserved field used for padding \*/

} INDUSTRYFILMINFOHEADER;

FilmMfgId stores the film manufacturer's ID code represented by the same two digits from the film edge code.

FilmType stores the film type code represented by the same two digits from the film edge code.

Offset stores the offset in perfs represented by the same two digits from the film edge code.

Prefix stores the prefix code represented by the same six digits from the film edge code.

Count stores the count code represented by the same four digits from the film edge code.

Format is a 32-byte ASCII string specifying the film format.

FramePosition specifies the frame number in the image sequence.

SequenceLen specifies the total number of frames in the image sequence.

HeldCount specifies the number of sequential frames to hold the current frame. This value is used to inject a run of identical frames into a motion picture sequence.

FrameRate stores the frame rate of the original film source in frames per second.

ShutterAngle specifies the shutter (temporal sampling aperture) angle of the motion picture camera in degrees.

FrameId is a 32-byte string identifying the type of frame (key frame, wedge frame, and so on). The format of the data in this field is defined by the creator of the DPX file.

SlateInfo is a 100-byte ASCII string that is used to store production information found on the camera slate.

Reserved is a 56-byte field used to pad the header out to 256 bytes in length. Future versions of the DPX format may support additional fields in this area.

#### Television Information Header

The second industry header contains information used by the television broadcast industry. The information in this header describes the video signal and television broadcast source from which the image data was derived. This header is 128 bytes in size and has the following format:

typedef struct \_IndustryTelevisionInfoHeader

{

DWORD TimeCode; /\* SMPTE time code \*/

DWORD UserBits; /\* SMPTE user bits \*/

BYTE Interlace; /\* Interlace \*/

BYTE FieldNumber; /\* Field number \*/

BYTE VideoSignal; /\* Video signal standard \*/

BYTE Padding; /\* Structure alignment padding \*/

SINGLE HorzSampleRate; /\* Horizontal sampling rate in Hz \*/

SINGLE VertSampleRate; /\* Vertical sampling rate in Hz \*/

SINGLE FrameRate; /\* Temporal sampling rate or frame rate

in Hz \*/

SINGLE TimeOffset; /\* Time offset from sync to first pixel \*/

SINGLE Gamma; /\* Gamma value \*/

SINGLE BlackLevel; /\* Black level code value \*/

SINGLE BlackGain; /\* Black gain \*/

SINGLE Breakpoint; /\* Breakpoint \*/

SINGLE WhiteLevel; /\* Reference white level code value \*/

SINGLE IntegrationTimes; /\* Integration time(s) \*/

BYTE Reserved[76]; /\* Reserved field used for padding \*/

} INDUSTRYTELEVISIONINFOHEADER;

TimeCode stores the SMPTE time code value for the image data.

UserBits stores the SMPTE user bits value for the image data.

Interlace is 0 if the image data is not interlaced and 1 if the data is stored using a 2:1 interlace.

FieldNumber indicates the field number to which the image data corresponds, if appropriate. Values are typically 1 or 2 for composite video, 1 to 4 for NTSC, or 1 to 12 for PAL. This value is 0 if the image data is not associated with a particular field.

VideoSignal specifies the video signal standard of the video source. There are 256 possible values to this field; the following are defined:

|  |  |
| --- | --- |
| 0 | Undefined |
| 1 | NTSC |
| 2 | PAL |
| 3 | PAL-M |
| 4 | SECAM |
| 50 | YCBCR CCIR 601-2 525-line, 2:1 interlace, 4:3 aspect ratio |
| 51 | YCBCR CCIR 601-2 625-line, 2:1 interlace, 4:3 aspect ratio |
| 100 | YCBCR CCIR 601-2 525-line, 2:1 interlace, 16:9 aspect ratio |
| 101 | YCBCR CCIR 601-2 625-line, 2:1 interlace, 16:9 aspect ratio |
| 150 | YCBCR 1050-line, 2:1 interlace, 16:9 aspect ratio |
| 151 | YCBCR 1125-line, 2:1 interlace, 16:9 aspect ratio |
| 152 | YCBCR 1250-line, 2:1 interlace, 16:9 aspect ratio |
| 200 | YCBCR 525-line, 1:1 progressive, 16:9 aspect ratio |
| 201 | YCBCR 625-line, 1:1 progressive, 16:9 aspect ratio |
| 202 | YCBCR 787.5-line, 1:1 progressive, 16:9 aspect ratio |

All values not defined are reserved for use in future versions of DPX.

Padding is a one-byte field used only to maintain the alignment of the structure elements on a 4-byte boundary.

HorzSampleRate and VertSampleRate specify the horizontal and vertical sampling rate of the video data in samples per second.

FrameRate specifies the temporal sampling rate, or the frame rate, in samples per second.

TimeOffset stores the time offset from sync (the reference edge of the horizontal sync) to the first pixel in microseconds.

Gamma is the gamma correction exponent value for the image. The default gamma value is 2.2 for an NTSC image.

BlackLevel specifies the digital value representing reference black. The default value for this field differs depending upon the video signal standard of the image data source.

BlackGain defines the linear gain applied to signals below the breakpoint threshold value.

Breakpoint defines the signal level threshold above which the gamma law is applied.

WhiteLevel specifies the digital value representing reference white. The default value for this field differs depending upon the video signal standard of the image data source.

IntegrationTimes indicates the temporal sampling aperture of the television camera.

Reserved is a 76-byte field used to pad the header out to 128 bytes in length. Future versions of the DPX format may support additional fields in this area.

Note that if the value of the VideoSignal field is 0 (undefined), the Gamma, BlackLevel, BlackGain, Breakpoint, and WhiteLevel values cannot be assumed; therefore, these fields must be initialized to their correct values.

### User-defined Data

The user-defined data following the industry headers is optional and is not present in a DPX file if the value of the UserSize field in the generic file header is FFFFFFFFh. Otherwise, the value of UserSize indicates the total size of the user-defined data in bytes.

User-defined data may include any information for which the DPX format does not provide storage. Such information includes thumbnail images, additional textual documentation, processing logs, verification signature, color maps, and so on.

User-defined data may be from 32 to 1,048,608 bytes in length and has the following format:

typedef struct \_UserDefinedData

{

char UserId[32]; /\* User-defined identification string \*/

BYTE \*Data; /\* User-defined data \*/

} USERDEFINEDDATA;

UserId is a string used to identify the data found in the Data field. This field is used by a DPX file reader to verify that it understands the format of the Data field.

Data is the actual user-defined data. The format of the field is not defined by the DPX standard and the data in this field may be up to one megabyte (1,048,576 bytes) in length.

### Image Data

The image data follows the headers and any user-defined data. It is recommended that the image data begin on an 8K block boundary, so there will be padding between the last header, or user-defined data, and the start of the image data. The ImageOffset value in the file header is used to locate the start of the image data.

The image data is stored as an array of 32-bit elements made up of four signed or unsigned character values. Because unsigned values are the default for DPX image data, we will represent the image data array here using the following type definitions:

typedef unsigned char BYTE4 [4];

typedef struct \_ImageDataElement

{

BYTE4 \*Data;

} IMAGEDATAELEMENT;

The DPX format stores image data by pixels, separating each pixel into its component values (also called *samples* in other formats) and storing each value in a separate element (also called a *color plane*). Image data components may be stored in up to eight elements total.

For example, an 8-bit gray-scale pixel contains only one component and is therefore stored as one component value only in one element. A 32-bit RGBA pixel contains four components that may be stored across four elements as four 8-bit component values or may be stored in a single element as one 32-bit component value.

As you can see, the DPX format is quite flexible in allowing you to store your image data in any way that makes reading and writing the data the most efficient for your computer hardware.

All components must be the same size across all elements. Valid component sizes are 1-, 8-, 10-, 12-, and 16-bit integers and 32- and 64-bit reals (IEEE floating-point). All components must be stored as words using 32-bit boundaries.

One-bit data is stored in a single element with the first (leftmost) pixel in the least significant bit in the byte and stored as 32-bit data. Eight-bit data is stored four components to a word, 16-bit as two components, 32-bit as one component, and 64-bit components require two 32-bit words to store. All of these component sizes fit evenly within 32-bit word boundaries.

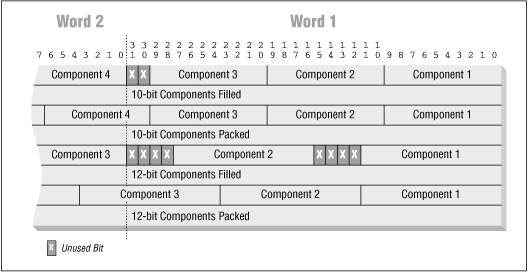
Ten- and 12-bit data, however, does not fit evenly within 32-bit boundaries. In these cases, 10- and 12-bit components are either "filled" into the 32-bit words, leaving unused bits in each word, or "packed" into 32-bit words, leaving no unused bits but causing components to be split across the 32-bit word boundaries.

Filling is accomplished by storing as many components as will fit within a 32-bit word and regarding the remaining bits as padding. For example, three 10-bit components will fit in a 32-bit word using bits 0:9, 10:19, and 20:29. The two remaining bits, 30:31, are not used and are set to zero. Two 12-bit components may be stored in a 32-bit word, but each component is stored starting on a 16-word boundary using bits 0:11 and 16:27. Bits 12:15 and 28:31 are not used for data and only provide alignment padding. One-, 8-, 16-, 32-, and 64-bit data fits evenly within 32-bit boundaries and is therefore never filled.

Packing is used to avoid wasted bits by disregarding all byte and word boundaries. In the previous example, we noted that three 10-bit components use the first 30 bits of a word and waste the last two bits. The fourth 10-bit component then begins on the boundary for the next 32-bit word. If these components were packed, the fourth component would begin on bit 30 of the first word, and would cross over the 32-bit boundary to occupy the first eight bits of the second word.

In the packed format, bits are always stored from the least significant to the most significant bits in a byte. When a component wraps around to the next 32-bit word, its remaining bits are placed in the least significant bit of that word. Packing always stops at the end of a scan line, and any remaining bits are set to zero. (See Figure DPX-2.)

#### Figure DPX-2: Packed bits



Lines are typically padded with zero bits to end on 4-byte boundaries, although this is not a requirement of the DPX standard. Image data is also typically padded with zero bits to end on an 8K block boundary.

### Data Compression

Components may be stored in an image element using a compressed or uncompressed format. Uncompressed component values are stored directly as raw data without any form of encoding. Optionally, components may be compressed using a simple run-length encoding scheme.

Runs of component values are encoded as a single flag value followed by one or more component values. The least significant bit of the flag value is one if the run of components values are identical, and zero if the run of component values are different. The remaining bits in the flag value specify the number of components in the run. The flag value is always the same size as the component values.

Runs of identical component values are encoded as a flag value followed by a single component value that specifies the value of the run. Runs of different component values are encoded as a flag value followed by a number of components equal to the component count stored in the flag value. A chrominance (CbCr) value is a single component stored as two values.

Component run encoding always stops at the end of scan lines. And encoded data is also packed or filled into 32-bit word boundaries, as indicated by the Packing field in the generic image header.

### DPX Core Requirements

The DPX standard provides for a minimally defined version of the DPX format that allows quick implementation of DPX file readers and writers, and defines a minimal set of core header fields that must be read and properly initialized to interpret the data stored in a DPX file.

The core header fields of a DPX file are the Magic, ImageOffset, Version, and FileSize fields of the generic file header, and the Orientation, NumberElements, PixelsPerLine, LinesPerElement, DataSign, Descriptor, Transfer, Colorimetric, BitSize, Packing, Encoding, and DataOffset fields of the generic image header.

Minimal DPX file readers must read these fields and may ignore all others. Minimal DPX file writers must initialize these core fields with valid values and may initialize all other header fields with undefined values.

Once all headers, optional user-defined data area, and image data have been defined, the following structure may be used to store the contents of a single DPX file:

typedef struct \_DpxFileFormat

{

GENERICFILEHEADER FileHeader;

GENERICIMAGEHEADER ImageHeader;

GENERICORIENTATIONHEADER OrientHeader;

INDUSTRYFILMINFOHEADER FilmHeader;

INDUSTRYTELEVISIONINFOHEADER TvHeader;

USERDEFINEDDATA \*UserData; /\* NULL if data not present \*/

IMAGEDATAELEMENT \*ImageData; /\* Array of 1 to 8

IMAGEDATAELEMENT structures \*/

} DPXFILEFORMAT;

## For Further Information

The DPX file format specification is available in the following document:

ANSI/SMPTE 268M-1994, *SMPTE Standard for File Format for Digital Moving-Picture Exchange (DPX)*, v1.0, 18 February 1994, 14 pages.

This document is available directly from SMPTE:

The Society of Motion Picture and Television Engineers  
595 W. Hartsdale Avenue  
White Plains, NY 10607-1824  
Voice: 914-761-1100  
FAX: 914-761-3115  
Email: *smpte@smpte.org*  
WWW: *http://www.smpte.org*

SMPTE is a professional society for motion picture and television engineers that is devoted to advancing the theory and application of motion-imaging technology, including film, video, television, computer imaging, and telecommunications.

The following paper is also a helpful source of information about the DPX format:

Snider, David, Glenn Kennel, Ken Curry, and Michael McCracken, "Digital Moving-Picture Exchange: File Format and Calibration," *SMPTE Journal*, August 1993, pp. 712-714.

The authors are all from the Eastman Kodak Co. in Rochester, New York. Some of them also served on the committee that prepared the SMPTE 268M standard.