FlashPix Format Specification

Version 1.0

For change requests, please send e-mail to “format_change_request@kodak.com”

A public listserver has been established for interested parties to share information and ideas regarding the *FlashPix* format. To subscribe to this listserver do the following:

1. Compose an e-mail message to “maillist@pixel.kodak.com”

2. The note should not contain a subject line and should have only one line in the body as follows:

subscribe FORMAT firstname lastname (i.e. subscribe FORMAT John Doe)

3. Whatever address you send this note from will be the address all listserver messages are sent to (i.e. john_doe@kodak.com)

4. A confirmation message will be sent back to you containing instructions on how to communicate with the listserver.

Contents

List of figures

List of tables

SECTION 1

Introduction

1.1 Purpose

This document is the technical specification that defines the image file format for *FlashPix* images. The effort to define the *FlashPix* format is a cooperative endeavor that includes the Eastman Kodak Company, Microsoft Corporation, the Hewlett-Packard Company, and Live Picture, Inc. The FlashPix format builds on the best features of existing formats (Kodak Image Pac, Live Picture IVUE, Hewlett-Packard JPEG, TIFF, TIFF/EP, and so on), and combines these features with an object orientation and other powerful new capabilities.The *FlashPix* format enables the industry to deliver desktop computer solutions that make it easy, enjoyable, and commonplace to use digital color photographic images in offices and homes.

1.2 Specification Organization

This document is divided into several sections, each isolating one aspect of the *FlashPix* specification. The sections are as follows:

- *Section 1: Introduction* defines structured storage.
- *Section 2: Image Data Representation* describes the resolution hierarchy and the organization of image data into tiles.
- *Section 3: The FlashPix Image Object* describes the format, in terms of Structured Storage, of the *FlashPix* image object.

- *Section 4: Image Data Format* describes the format of the individual streams used to store the actual image data.
- *Section 5: Color Space Specifications* defines the PhotoYCC and NIF RGB color spaces.
- *Section 6: Image Info Property Set* describes the non-image information in a *FlashPix* image.
- *Section 7: FlashPix Image View Object* defines the *FlashPix* image view object. This object allows a default view (including orientation, crop and color adjustment) to be specified for a *FlashPix* image without modifying pixel values.
- *Appendix A: Structured Storage* defines the binary format of structured storage files and relevant data structures.

1.3 Conventions

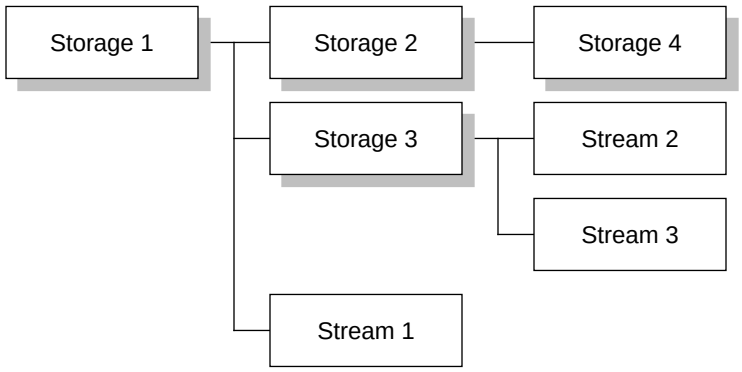
The following conventions are used in this document.

- All numbers starting with a "0x" are in hexadecimal.
- Special characters in strings are indicated using the octal format, where any three digit number preceded by a "\" represents the ASCII value of the character in octal. For example, a newline character would be represented by "**\012**."
- Spaces in strings are explicitly indicated using their octal representation, "**\040**."
- Stream and storage names are specified in standard C language "printf" syntax.

1.4 Structured Storage

The *FlashPix* format is based on a compound object storage model called structured storage. A file in structured storage format contains two types of objects: storages and streams. Storages are analogous to directories in a file system; streams are analogous to files. A storage may contain both zero or more additional storages and zero or more streams. The streams and storages in a *FlashPix* file are individually addressable. Figure 1.1 shows the convention used in this document to illustrate storages and streams.

FIGURE 1.1 **Conventions for storages and streams in illustrations**



The entire structured storage file appears in the host file system as one file. In this example, storage 1 represents the root storage. It is the highest level storage of the file and is the entity that is visible in the host file system. It contains two storages (2 and 3) and one stream (1). Storage 2 contains one empty storage (4). Storage 3 contains two streams (2 and 3).

All entities in a *FlashPix* file have a class ID that identifies the type of the object. Class IDs are defined as globally unique identifiers (GUID's). They are represented as 128 bit numbers that are considered impossible to duplicate. GUID's are generated using the algorithm specified for the generation of universal unique identifiers for remote procedure calls[17].

1.4.1 Property Sets

Structured storage defines property sets as a stream for storing tagged data. The *FlashPix* format uses this mechanism extensively for storing data other than actual pixel values.

As defined, property sets are very flexible. All property sets must be in Windows Format. Windows format is indicated in the property set header by setting the wByteOrder field to 0xFFFE and the wFormat field to 0x0. Furthermore, the codepage must be written into the requisite property (PID = 1) in each and every FlashPix property set as described below. A binary specification of property sets is include in this specification in Section A.2.

With the sole exception of the OLE standard Summary Information Property Set each and every *FlashPix* property set must be in the Unicode (1200) codepage, and all strings in that set must be stored as wide 16-bit characters (LPWSTR). Due to its origin and use in non-*FlashPix* applications, the Summary Information Property set has different conditions than all other property sets. This property set must be in the Western European ANSI (1252) code page, and all strings in it must be stored as 8-bit characters (LPSTR). For legacy reasons, all *FlashPix* property set readers must be able to handle a 1200 codepage in the Summary Information Property set as if it was 1252. (Note - this

last restriction will not introduce any character shifts since the 8-bit subsection of the Unicode codepage is exactly the 1252 codepage.)

The properties defined for each property set are listed in the property set definition. All property ID codes not explicitly listed for the property set are reserved for registered extensions. Where valid property values are listed, those not explicitly listed are reserved for registered extensions.

1.4.2 Summary Information Property Set

Stream name: **\005**SummaryInformation
Class ID: F29F85E0-4FF9-1068-AB91-08002B27B3D9
Format ID: F29F85E0-4FF9-1068-AB91-08002B27B3D9

Structured storage defines one property set that can be found in every *FlashPix* object to provide a basic level of information about the object. This summary information property set is used in *FlashPix* image objects and *FlashPix* image view objects. This property set is in general use in OLE. It must be in the Western European ANSI (1252) codepage. Because of this restriction, this property set is not truly internationalizable in version 1.0 of the *FlashPix* specification.

The property set must also have at least one section that has a format ID the same as the class ID. The properties of the summary information property set are listed in Table 1.1.

Title property (optional)
This property is available for the application to record the object title.

Subject property (optional)
This property is available for the application to record the subject of the object.

Author property (optional)
This property is available for the application to record the author of the object.

Keywords property (optional)
This property is available for the application to record keywords about the object.

Comments property (optional)
This property is available for the application to record comments about the object.

Template property (optional)
This property is not used with *FlashPix* objects.

Last saved by property (optional)
This property is available for the application to record the name of the user who last saved the object.

Revision number property (optional)
This property is available for the application to record the number of times the object has been saved.

TABLE 1.1 **Valid properties of the summary information property set**

Property Name	ID Code	Type
Title	0x00000002	VT_LPSTR
Subject	0x00000003	VT_LPSTR
Author	0x00000004	VT_LPSTR
Keywords	0x00000005	VT_LPSTR
Comments	0x00000006	VT_LPSTR
Template	0x00000007	VT_LPSTR
Last saved by	0x00000008	VT_LPSTR
Revision number	0x00000009	VT_LPSTR
Total editing time	0x0000000A	VT_FILETIME
Last printed	0x0000000B	VT_FILETIME
Create time/date	0x0000000C	VT_FILETIME
Last saved time/date	0x0000000D	VT_FILETIME
Number of pages	0x0000000E	VT_I4
Number of words	0x0000000F	VT_I4
Number of characters	0x00000010	VT_I4
Thumbnail	0x00000011	VT_CF
Name of creating application	0x00000012	VT_LPSTR
Security	0x00000013	VT_I4

Total editing time property (optional)
This property is available for the application to record the duration of an object editing session.

Last printed property (optional)
This property is available for the application to record when the object was last printed.

Create date/time property (optional)
This property is available for the application to record the creation date and time for the object. This value should not be updated after it is initially written.

Last saved date/time property (optional)
This property is available for the application to record the date and time that the object is saved. It is strongly recommended that this property be used in *FlashPix* objects.

Number of pages property (optional)
This property is not used in *FlashPix* objects.

Number of words property (optional)
This property is not used in *FlashPix* objects.

Number of characters property (optional)

This property is not used in *FlashPix* objects.

Thumbnail property (required in some situations)

This property is available for the application to record a small bitmap representation of the *FlashPix* object. For the *FlashPix* image view object, the thumbnail property is optional for the summary information property set if this *FlashPix* image view points to a non-hierarchical source image object written in an embedded capture environment. The thumbnail is otherwise required for *FlashPix* image view objects. The thumbnail is optional for the source and result *FlashPix* image objects.

- Thumbnail data should reflect image contents within the thumbnail format limits and must be oriented the same way as the object it is contained in. Refer to Sections 3.1.2 and 7.1 for additional information about how the thumbnail property is used.
- The thumbnail image is stored in CF_DIB format which is a simple rectangular array of pixels with a small header as defined in [23].
- For single channel images (including opacity only images), treat them as monochrome without an opacity channel for purposes of the thumbnail.
- For multicolored images, all pixels are stored in 24 bit (bi.BitCount = 24) BGR format (in the NIF RGB color space). For single channel images, all pixels are stored in 8 bit (bi.BitCount = 8) format.
- Palettized color representations are not allowed for 24 bit DIB's. However, for single channel thumbnails, a palette entry must be provided which serves as the 8 to 24 bit identity lookup table. It is highly suggested that this palette be a pure grayscale ramp of exactly 256 RGBQUAD elements (e.g. biClrUsed = 0) running from black to white. The palette should consist of a sequence of 256 32-bit RGBQUAD structures [x,x,x,0] for all x running from 0 to 255. Note that DIB palettes require the fourth (reserved) channel to be identically zero as defined in [23].
- The thumbnail image data is stored uncompressed.
- For images with an opacity channel in addition to image data channels, the thumbnail should be stored as if it had been composited on a fully opaque white background.
- The larger of the thumbnail stored height and width must be 96 pixels. The image should be resized to this dimension instead of padding a smaller image. It is not required to pad the smaller dimension to 96 pixels.

Name of creating application property (optional)

This property is available to the application to record the name of the application that created the object. It is strongly recommended that this property be used in *FlashPix* objects.

Security property (optional)

This property is not used in *FlashPix* objects.

1.4.3 File identification

A *FlashPix* file must be an image view object and its object type class ID must be stored in the root storage header. Many object based systems (e.g. OLE) will use the class ID found in the header of the root storage as a key for launching an application. In this way an application can be designated to handle all files of this object type by default, regardless of their creator.

The *FlashPix* image view object and *FlashPix* image object storages are required to have a CompObj type stream. Their object type class ID is required to be stored in the clipboard format field of that stream as well as in the header of the storage. The clipboard format field is what should be used to determine the class ID of these objects.

In non-OLE environments, *FlashPix* format files are identified by other means; including file name extensions and file types.

Macintosh systems use a file type designation to identify file content [24]. The file type is a 4-character code stored with each file. For example, the code 'TEXT' indicates that the file contains ASCII text. The Macintosh also associates a "file creator" with each file. This is also a 4-character code. It identifies the application that created the file.

These two codes are stored in the Finder Desktop Database [24]. When a file is double-clicked, the Finder uses the file's creator ID to determine its associated application. On the Macintosh, *FlashPix* files have been registered with a file type of "FPix" (Hex) 46506978.

On platforms (e.g., UNIX) that do not support GUIDs or file types as a means of associating files and applications/components, file extensions should be used. These are specified by the user as a period (".") followed by up to 3 letters. For example, an extension of (".FPX") can be used to indicate that the file contains a *FlashPix* file.

1.4.4 OS-level file treatment in Windows or with OLE

It is recommended that in Windows or OLE-enabled environments, core *FlashPix* files be managed independent of their creating application and that the user receives some control over which *FlashPix* reader becomes the server for *FlashPix* files.

The former can be accomplished by writing and saving core *FlashPix* files using the class ID of the *FlashPix* object type in the header portion of the root storage. Such files should also use a ‘.FPX’ extension. Cases where it is more appropriate to use the creating application’s class ID as the root storage class ID include: images with significant use of an application’s extensions, user approval via explicit prompting or preference setting, or for images used in a closed system.

In OLE-enabled environments, core *FlashPix* reader applications should be able to register as the server application for the *FlashPix* image view object class ID. If all core *FlashPix* readers did this, the last installed application would become the default server application. Due to the confusion this can cause the user, it is strongly recommended

that the installation procedure for a core *FlashPix* reader application offer it as the default *FlashPix* application. The installation would not insert the new application as the server of the *FlashPix* image view object class ID unless the user agreed. Whenever possible, the installation procedure would also register the application as serving the .FPX extension.

1.4.5 *FlashPix* Streams

FlashPix class IDs are used to identify the type of an entity. Unfortunately, standard structured storage streams do not have a class ID. To deal with this problem a *FlashPix* stream differs from an OLE stream in that the first 28 bytes of the actual stream contain header information, part of which is the class ID of the stream. These bytes are not counted when determining offsets into the stream nor when determining the stream length. For example, the first byte of data in a *FlashPix* stream is stored in byte 28 of the actual stream (the first byte is byte 0). These 28 bytes are in the format of a property set header, as defined in Section A.2.1.1.

Note that property sets, on the other hand, do have a class ID field, and thus do not need modification. Property sets in the *FlashPix* format are not stored as *FlashPix* streams, but as standard streams. However, the header of a *FlashPix* stream is the same as the header of a property set.

1.4.6 String and Character Representation

There are numerous fields in the *FlashPix* format that contain character strings. These may be broken down into two general classes: structured storage related names such as stream and storage pathnames, and descriptive strings stored in property sets such as image title, film type and keywords.

To promote the ability to transfer *FlashPix* images internationally, nearly all strings in the *FlashPix* format are stored in the Unicode format. However, to promote the ability to transfer *FlashPix* files among different operating systems, all structured storage related names are required to be in the 7-bit ASCII compatible part of Unicode. Descriptive strings are allowed a much wider range, since they can be ignored outside the language they were generated in.

1.4.6.1 Storage and Stream Names

The *FlashPix* file is built on top of structured storage files, which in turn are built on top of the host file system. Hence, it must follow the conventions placed on stream and storage names, and the file system names.

Each structured storage file has a single root storage. The name of the file root storage may be any valid storage name. However, it is recommended that the file name in the host file system be used.

Names of *FlashPix* streams contained within storage objects are managed by the implementation of the particular storage object in question. Names are stored case-preserving,

but are compared case-insensitive. As a result, all *FlashPix* writers which define storage and stream names must choose names which will work in either situation. *FlashPix* streams and storage names may be up to 31 characters in length.

Although storage and stream names are actually stored as 16-bit Unicode characters, the characters must be within the lower 7-bit ASCII range with the following additional restrictions:

- The names "." and ".." are reserved for future use.
- The four characters "\", "/", ":", or "!" are not allowed.

Restricting such string names to 7-bit ASCII greatly promotes interoperability across different platforms without significantly impacting internationalization, as these stream and storage names are rarely exposed to the user.

In addition, the name space in a storage is partitioned into different areas of ownership. Different pieces of code have the right to create elements in each area of the name space:

- Storage and stream names beginning with characters "**\001**" through "**\004**" are reserved for OLE.
- Property set names must begin with the "**\005**" character.
- Storage and stream names beginning with character "**\006**" through "**\037**" are reserved for OLE or for future use.
- Any other character may be used to begin a storage or stream name.

1.4.6.2 Property Set Code Page and Strings

The other major area string names kept in the *FlashPix* format are in property sets. Property sets are defined with a "code page" specifying the type of characters allowed, and numerous strings. All strings in the property set share the same code page.

All property sets in the *FlashPix* format must belong to the Unicode code page, as specified in Section 1.4.1. The code page ID must be stored in property ID 0x00000001 in all *FlashPix* property sets. These property sets are defined in the rest of this document. Hence it is an error to not specify the code page, or to specify it with any other value than 0x04B0, in any property set in a *FlashPix* file, as described in Section A.2.2.2. However, applications may write application specific property sets in a *FlashPix* file. Only the stream names for the private extensions are not required to be in the Unicode code page, although this is strongly recommended. Note that the count at the start of VT_LPWSTR is to be interpreted as a character count, not a byte count. The count includes the null character at the end of the string.

Descriptive strings such as names, camera types, and subjects are not required to be within the 7-bit ASCII subset of Unicode. They are more often displayed and manipulated by the user, and it is a requirement that the storage of the full character set is supported.

However, it is not required that a non-internationalized application be able to display and manipulate characters outside the ASCII and local character set. It is perfectly

acceptable for a German *FlashPix* reader/writer, for instance, to display question marks or similar symbols if passed Kanji characters. However, it is required that all Unicode characters be preserved on copying. If that string was not modified by the German user, it must remain readable under a Japanese version of the application.

String properties may be stored as either 8-bit strings (VT_LPSTR) or 16-bit strings (VT_LPWSTR). Note because of Unicode compatibility, the upper 128 elements in an 8-bit string fall into the Western-European 8-bit character range in the Unicode code page[20]. Hence, strings in those languages may be adequately stored as 8-bit strings. Characters from other languages (Greek, Russian, Japanese, for example) must be stored in 16-bit strings.

All *FlashPix* property set readers must be able to read either 8 or 16 bit (Unicode) strings for any property. This imposes a requirement that all property set readers be able to convert 8 or 16 bit strings into the desired internal format for the application. It is advised that 16-bit strings be utilized whenever possible.

1.5 Format Compliance

The *FlashPix* image format has a core definition and defined extensions. *FlashPix* files, reader software, and writer software must be compliant with the core definition and may optionally support one or more extensions.

The core *FlashPix* format definition specifies the required and optional data elements and allowed data values that compose *FlashPix* files and default actions of *FlashPix* reader software:

- Core *FlashPix* files must contain all required core *FlashPix* data elements and any of the core *FlashPix* optional data elements using only those values enumerated in the core *FlashPix* definition.
- Core *FlashPix* reader software must read all valid core *FlashPix* file permutations and take all default actions defined in the core *FlashPix* specification.
- Core *FlashPix* writer software must write at least one valid core *FlashPix* file permutation.

Extensions to the core *FlashPix* format may be defined to add features that are not supported in the core *FlashPix* definition. Each extension must be defined in a way which does not prevent core *FlashPix* reader software from productively interpreting *FlashPix* files with the extension present. Core *FlashPix* reader software that doesn't support a particular extension will ignore its added data elements and will resort to default actions when non-core values are present in core data elements.

Extensions to the *FlashPix* format are characterized by the feature capability being added and defined by the required and optional data elements and values associated with the feature. An extended *FlashPix* file meets the definition of core *FlashPix* files, but has additional data elements for the extensions present in the file. Extended *FlashPix*

reader software is core *FlashPix* reader software with additional capability to productively interpret one or more defined *FlashPix* extensions. Extended *FlashPix* writer software is core *FlashPix* writer software with additional capability to create the data elements associated with one or more defined *FlashPix* extensions.

FlashPix extensions may either be registered or private. Registered extensions are collaboratively defined via the the *FlashPix* format Advisory Council and published publicly as separate documents from the core *FlashPix* format specification. Private extensions are defined by any interested party and shared at their discretion.

1.6 *FlashPix* File Overview

A core *FlashPix* file is composed of a *FlashPix* image view object which contains scriptable image transforms, a source *FlashPix* image object, and, optionally, a result *FlashPix* image object containing the result of applying the transforms to the source *FlashPix* image object. The *FlashPix* image object is defined in *Section 3: The FlashPix Image Object* and the remainder of the *FlashPix* image view object is defined in *Section 7: FlashPix Image View Object*. Those storages and streams in italics are optional or optional under specific circumstances.

Figure 1.2 is an overview of a *FlashPix* file, a *FlashPix* image view object. Figure 1.3 describes the content of each *FlashPix* image object in the *FlashPix* image view object and Figure 1.4 describes the content of each resolution storage in the *FlashPix* image objects.

FIGURE 1.2 ***FlashPix* image view object**

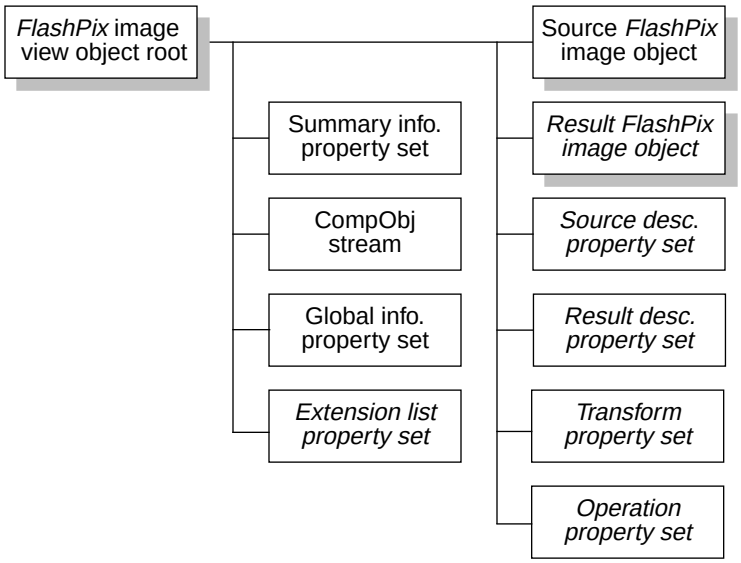


FIGURE 1.3 ***FlashPix* image object**

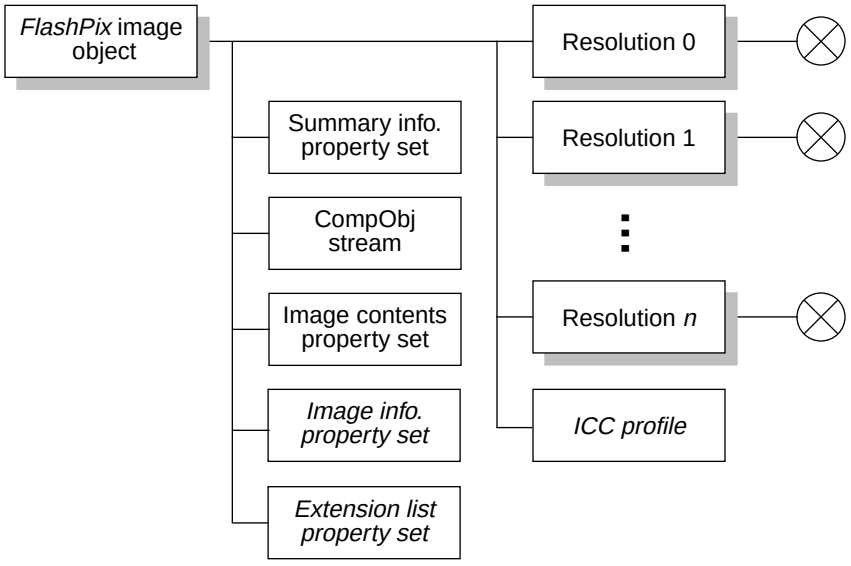
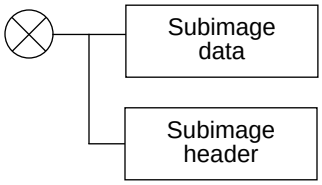


FIGURE 1.4 **Contents of a resolution storage**



1.6.1 Extension management

Both the *FlashPix* image view object and the *FlashPix* image objects may be independently extended. The extensions associated with each object are recorded in that object's extension list property set. Extensions to the *FlashPix* image view object may be located anywhere within the *FlashPix* file, even within a *FlashPix* image object. It is, however, not recommended that a *FlashPix* image view object extension place all of its data elements within a *FlashPix* image object. Extensions to a *FlashPix* image object must be entirely within the *FlashPix* image object.

FlashPix reader software must be able to interpret the extension list property sets and may also be an extended reader with the capability to interpret specific extensions which may be present in extended objects.

A particular *FlashPix* extension may not be valid if the core *FlashPix* data elements are edited. Each extension must have its persistence defined by the authoring application. The extension is marked as either: valid independent of any edits, invalid upon edits, or potentially invalid upon edits.

When opening a *FlashPix* file, the reader software should determine if there are any extensions present in the file which it does not support and which will become invalid or potentially invalid upon editing. If there are such extensions in the file, the user should be informed that the file is extended and that edits to the file may cause the loss of some of the extensions. For each extension that the reader software supports and which is marked as potentially invalid upon edits, the modification date of the extended object (in its summary info property set) must be compared to the extension modification date. If the extended object's modification date is more recent than that of the extension, the reader must determine if the extension is still valid. If it is not, it must either be updated or all of its data elements listed in the extension list property set must be deleted.

SECTION 2

Image Data Representation

2.1 Coordinate systems

FlashPix files store image data in a hierarchy of resolutions from the highest available for an image down to the lowest defined in the format. Image editing operations need to be supported within an individual resolution, but must also be applied to all other resolutions.

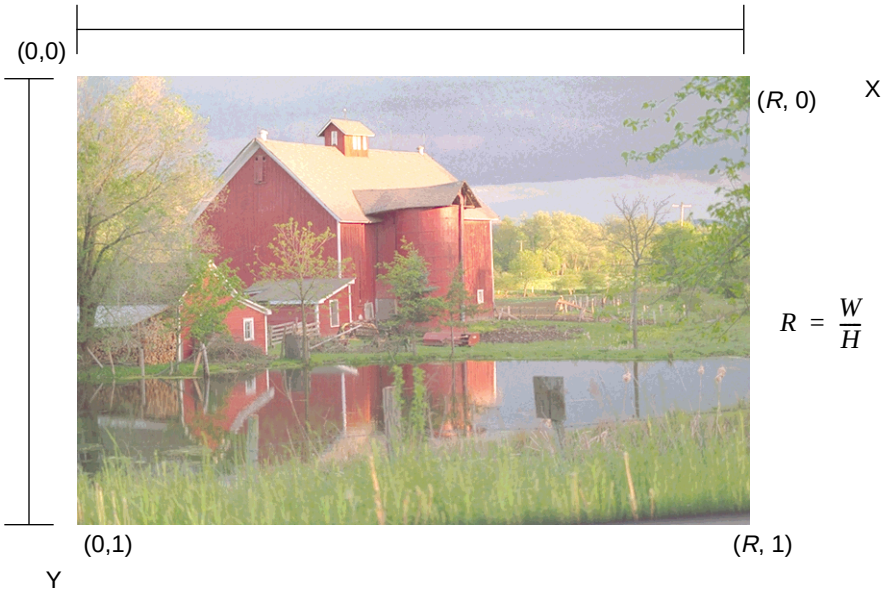
Therefore, two different coordinate systems are defined: resolution-independent and resolution-dependent.

2.1.1 Resolution-Independent Coordinates

In some situations, the image must be described by a coordinate system independent of the pixel.

Figure 2.1 shows a resolution-independent coordinate system. The image is described in a Cartesian system, with the X-axis horizontal and pointing to the right, the Y-axis vertical and pointing downward, and the origin at the upper left corner. The scale is such that the height of the image is normalized to 1.0. To keep the scale of the X-axis and the Y-axis the same, the image width is its aspect ratio (width/height). Thus, a square part of any image has equal width and height in this coordinate system.

FIGURE 2.1 **Resolution-independent coordinates**

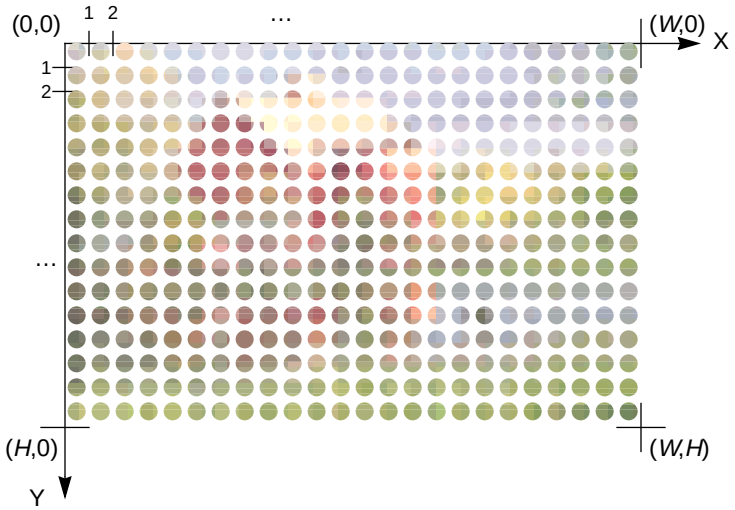


2.1.2 Resolution-Dependent Coordinates

At a given resolution, the normalized coordinate system described above must be converted to a set of discrete pixels. Then the continuous resolution-dependent coordinate system in Figure 2.2 is used. This is simply a scaled version of the previous coordinates. The values (x, y) in this coordinate system are still real (floating point) numbers.

To define the actual pixels of the image, an integer grid is overlaid on the coordinate system. The discrete pixel referred to by (i, j), where i and j are integers, is centered at location $(i+0.5, j+0.5)$. The half-unit shift makes the conversion between discrete and continuous descriptions simple. The point (x, y) falls in the unit square labelled $(\lfloor x \rfloor, \lfloor y \rfloor)$ and containing the pixel at $(\lfloor x \rfloor +0.5, \lfloor y \rfloor +0.5)$. No rounding is required.

FIGURE 2.2 **Resolution-dependent coordinates**



2.2 Multiple resolutions

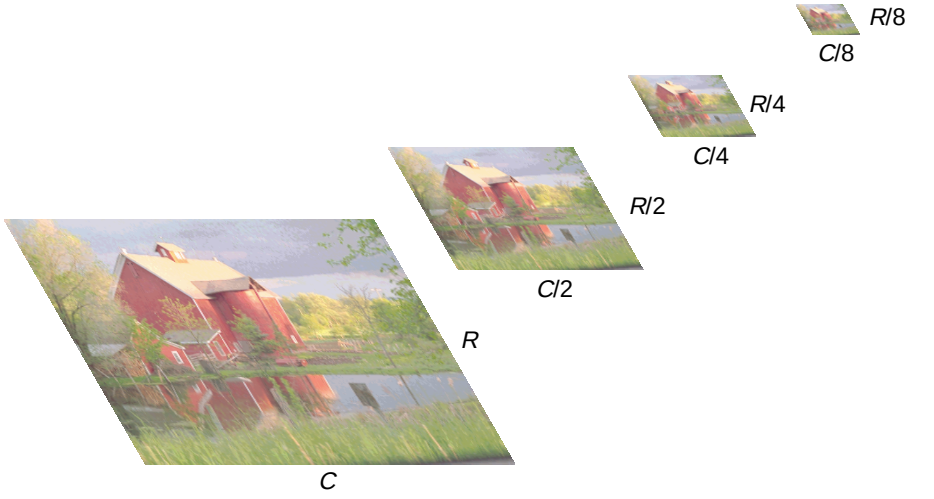
A *FlashPix* file must contain either a single resolution or the entire multi-resolution hierarchy. Each resolution in the full hierarchy is separated from the next higher resolution version by a spatial factor of 2× in both the *x* and *y* directions.

The series of resolutions continues until both the width and height of the smallest resolution are less than or equal to the width and height of a tile, 64 pixels. In the Figure 2.3 example, the tile width is smaller than *R*/4 but no smaller than *R*/8.

In Figure 2.3, the full resolution image is *R* rows × *C* columns. The actual spatial resolution (in pixels per inch, for example) is irrelevant, since neither the desired output size nor the output resolution is known. Each successively smaller resolution has half the number of rows and columns as the previous resolution. In this example, the second resolution is *R*/2 rows × *C*/2 columns.

Note that the *FlashPix* format uses centered subsampling. The pixels in resolution *i*-1 fall between the pixels in resolution *i*.

FIGURE 2.3 **Sample resolution hierarchy**



2.2.1 Resolution sizes

The size of a decimated image is determined from equation 2.1, where (w_0,h_0) is the width and height of the larger resolution and (w_1,h_1) are the width and height of the smaller resolution:

$$(w_1, h_1) = \left(\left\lfloor \frac{(w_0 + 1)}{2} \right\rfloor, \left\lfloor \frac{(h_0 + 1)}{2} \right\rfloor \right) \quad (2.1)$$

Note that this rounding affects the size of the image in resolution independent coordinates. The height of the largest resolution image is defined to be 1.0. Using the rounding method in Equation 2.1, the height of one resolution given the height of the next largest resolution can be determined as follows, where h_0 is the height of the larger resolution in resolution independent coordinates, p_0 is the height of the larger resolution in pixels, h_1 is the height of the next smaller resolution in resolution independent coordinates, and p_1 is the height of the next smaller resolution in pixels, as defined by Equation 2.1:

$$h_1 = \frac{2p_1}{p_0} \times h_0 \quad (2.2)$$

Failing to make this correction to the height and width of the image (in resolution independent coordinates) when dealing with resolutions other than the largest resolution may cause slight errors in the alignment of the multiple resolutions of the image.

2.2.2 Non-Hierarchical *FlashPix* Images

This specification also defines a non-hierarchical version of *FlashPix* images. In some cases, such as in the case of a digital camera, the system has neither the computing power to generate the full hierarchy nor the space to store the additional resolutions. In these cases, only the full-resolution image is stored. However, when this image can be used in an interactive manner, the full hierarchy must be constructed. This may happen in an acquire module accessing the device, in a separate converter program, or when an interactive core reader application opens the image and finds that it's hierarchy does not exist. A result image object, as defined in Section 7.1, also must be hierarchical when used in an interactive environment.

Non-hierarchical *FlashPix* images differ from fully hierarchical *FlashPix* images in that there is only one resolution. This distinction is specified in Section 3.1.5.1.

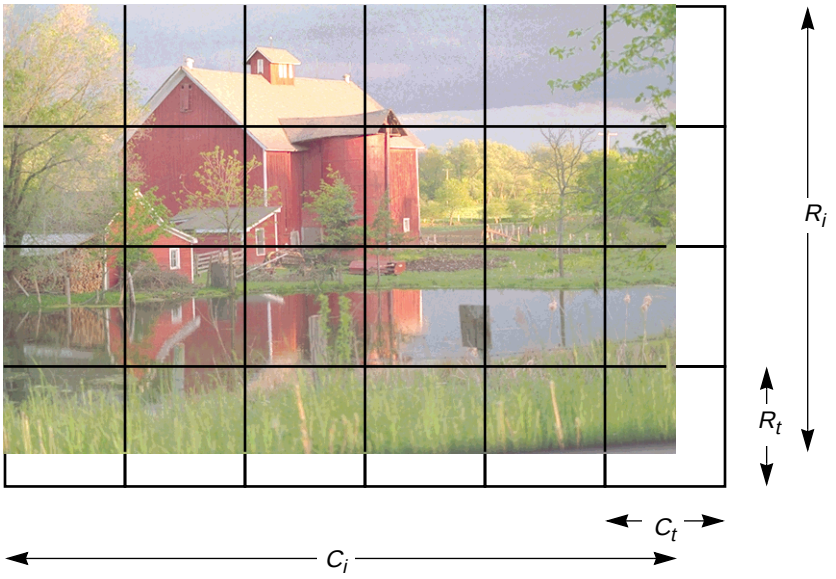
2.3 Tiling

In addition to providing the image at several resolutions, each resolution image is organized into tiles to provide more efficient access to any portion of the image.

2.3.1 Breaking an Image into Tiles

Figure 2.4 shows an image divided into tiles.

FIGURE 2.4 **A tiled image**



The example image is R_i rows × C_i columns, organized into R_t row × C_t column tiles. For any image, the number of tiles per row (N_R) and the number of tiles per column (N_C) are:

$$N_R = \left\lceil \frac{R_i}{R_t} \right\rceil \qquad N_C = \left\lceil \frac{C_i}{C_t} \right\rceil \tag{2.3}$$

For the image shown in Figure 2.4, $N_R = 4$ and $N_C = 6$.

However, it is unlikely that the image size will be a multiple of the tile size. As illustrated in Figure 2.4, the tiles on the right and bottom of the image will only partially contain valid image data. Since only full tiles can be stored in a *FlashPix* image, incomplete tiles must be padded to the full tile width and height. Tiles should be padded with values extruded from the image itself. For example, pixels to the right of the image should be padded with the value of the rightmost pixel in each row. Pixels specifying opacity data should be padded just as if they were image data, as the actual image size is specified by the actual width and height, not by the opacity data.

To access a particular pixel, the tile containing that pixel must first be located. To calculate the tile, described by the coordinate pair (T_C, T_R), where the upper left tile is (0, 0) and the lower right tile is (N_C - 1, N_R - 1), containing the pixel (c+0.5, r+0.5), use the following formulas:

$$T_R = \left\lfloor \frac{r}{R_T} \right\rfloor \qquad T_C = \left\lfloor \frac{c}{C_T} \right\rfloor \tag{2.4}$$

The tiles are numbered sequentially, starting at the upper left tile and proceeding from left to right and top to bottom. The number of the tile (T_C, T_R), T, is given by the following formula:

$$T = T_R \times N_C + T_C \tag{2.5}$$

Once the correct tile has been located, the location (c', r') of the unit square containing the desired pixel, with respect to the tile, can be determined:

$$r' = r \,\%\, R_T \qquad c' = c \,\%\, C_T \tag{2.6}$$

where ‘%’ represents the modulus operator.

SECTION 3

The FlashPix Image Object

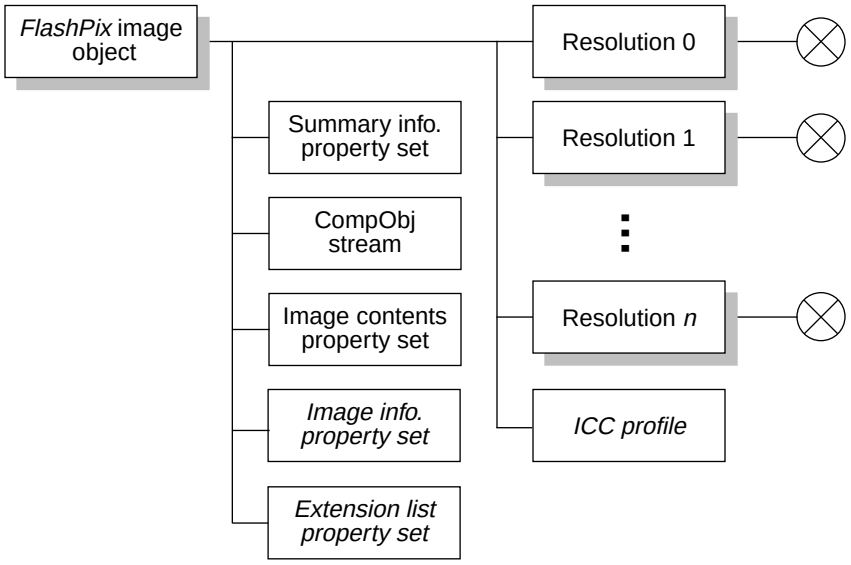
3.1 *FlashPix* Image Object Structure

The *FlashPix* image object is a storage whose contents are together treated as an image in the *FlashPix* format. Figure 3.1 shows the storages and streams in a *FlashPix* image object.

Storage name:	Data**040**Object**040**Store**040**%06d
Class ID:	56616000-C154-11CE-8553-00AA00A1F95B

The single numeric parameter in the storage name represents the index of the image as described in Section 7.1.2. This class ID is to be used for all *FlashPix* image objects whether or not they contain any extensions.

FIGURE 3.1 ***FlashPix* image object storages and streams**

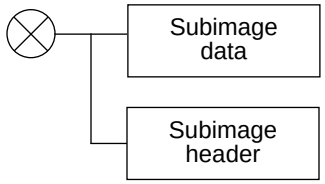


3.1.1 Resolution Storages

Storage name: Resolution**040**%04d
Class ID: 56616100-C154-11CE-8553-00AA00A1F95B

The decimal parameter in the storage name is the resolution number. Resolutions are numbered in sequence starting at 0, which is the lowest resolution.

FIGURE 3.2 **Contents of a resolution storage**



This series of storages contains the image data for each resolution. Storage contents are the same for every resolution, as implied by the connection point in Figure 3.1 and Figure 3.2.

Future extensions to the *FlashPix* format may include additional subimages, but in the core definition, only one subimage is allowed. The format includes provisions for multiple subimages, which is accomplished by treating the one subimage in the core definition as subimage 0.

Each resolution storage contains a header stream and a data stream that contain all the image data for that resolution. (*Section 4: Image Data Format* describes the format of the header and data streams.) The subimage must be in the same color space and numerical format for all resolutions.

3.1.2 Summary Info Property Set (required)

This property set is an instance of the standard Summary Information property set, as described in Section 1.4.2. This property set must adhere to the definition specified in Section 1.4.1.

The thumbnail property value, if defined, must be representative of the image. Therefore, in the result instance of the image object, the thumbnail must have had all transforms applied to it.

3.1.3 CompObj Stream (required)

The CompObj stream is a standard Structured Storage stream and is not a *FlashPix* stream. The header of the stream is not extended for storage of a stream class ID. This stream is required and is defined in Section A.3. The Unicode versions of the CompObj stream fields are required.

The CompObj stream Clipboard Format field is used to store the class ID of the *FlashPix* image object. The *FlashPix* image object class ID is converted to a string for storage in the Clipboard Format field and must be bracketed by the bracket characters ‘{‘ and ‘}’ just as returned by the OLE function StringFromGUID2.

The CompObj stream User Type field is generally used to store the User Type information from the OLE registry for the class ID. In OLE-enabled environments, the string contents should be retrieved from the OLE registry. In non-OLE-enabled environments, a string which is a user-understandable brief description of the object contents should be used.

The CompObj stream ProgID field is generally used to store the ProgID information from the OLE registry for the class ID. In OLE-enabled environments, the string contents should be retrieved from the OLE registry. In non-OLE-enabled environments, a string which identifies the program associated with the class ID should be used. This string cannot contain any spaces.

3.1.4 Image Info Property Set (optional)

Stream name: **\005**Image**040**Info
Class ID: 56616500-C154-11CE-8553-00AA00A1F95B
Format ID: 56616500-C154-11CE-8553-00AA00A1F95B

The image info property set contains properties that describe the actual image. (See *Section 6: Image Info Property Set* for a definition of this property set.)

3.1.5 Image Contents Property Set (required)

Stream name: **\005**Image**040**Contents
Class ID: 56616400-C154-11CE-8553-00AA00A1F95B
Format ID: 56616400-C154-11CE-8553-00AA00A1F95B

The image contents property set contains properties that describe how the image data is stored. The properties may appear in any order, but they are conceptually divided into three groups. The *primary description group* describes the *FlashPix* image as a whole, specifying the number of resolutions, the size of the largest resolution, etc. The *resolution description group* describes the subimage at each resolution. The *compression description group* contains image compression information.

The image contents property set is stored in standard property set format, adhering to the restrictions in Section 1.4.1. All property ID codes not explicitly listed are reserved for registered extensions.

3.1.5.1 Primary description group

This group contains properties (Table 3.1) describing the *FlashPix* image object as a whole.

TABLE 3.1 **Valid properties in the primary description group**

Property name	ID Code	Type
Number of resolutions	0x01000000	VT_UI4
Highest resolution width	0x01000002	VT_UI4
Highest resolution height	0x01000003	VT_UI4
Default display height	0x01000004	VT_R4
Default display width	0x01000005	VT_R4
Display height/width units	0x01000006	VT_UI4

Number of resolutions property (required)

This property specifies the number of resolutions contained in the *FlashPix* image. This property is required. The value must be the number of resolutions in the fully populated hierarchy or one for a non-hierarchical *FlashPix* image. That single stored resolution must be the highest resolution image. Its resolution number must be the number that would be assigned to the highest resolution image if the hierarchy were fully populated

(number of resolutions property value - 1 since lowest resolution is stored as resolution 0). The number of resolutions stored for a fully populated hierarchy can be calculated as shown in Equation 3.1, where tile size is 64.

$$\left\lceil \log_2 \frac{max(width, height)}{TileSize} \right\rceil \tag{3.1}$$

Highest resolution width and height properties (required)
These properties specify in pixels the height and width of the highest resolution image. Values do not include the padded area if the image data is padded to tile boundaries.

Default display height and width properties (optional)
These properties specify the default height and width for displaying the image.

Display height/width units property (optional)
If the default display height and width properties are present, this property is used to define their unit of measurement. Legal values are listed in Table 3.2. If this property is not present, *FlashPix* reader software must treat the image as though the value were Inches (0x0).

TABLE 3.2 Legal display height/width units property values

Value	Meaning
0x0	Inches
0x1	Meters
0x2	Centimeters
0x3	Millimeters

3.1.5.2 Resolution Description Groups

These groups, one for each stored resolution, contain properties to describe the subimage at that resolution. Table 3.3 lists the properties, where "*ii*" in the ID code is the resolution number.

TABLE 3.3 Valid properties in a resolution description group

Property name	ID code	Type
Subimage width	0x02*ii*0000	VT_UI4
Subimage height	0x02*ii*0001	VT_UI4
Subimage color	0x02*ii*0002	VT_BLOB
Subimage numerical format	0x02*ii*0003	VT_UI4 | VT_VECTOR
Decimation method	0x02*ii*0004	VT_I4
Decimation prefilter width	0x02*ii*0005	VT_R4
Subimage ICC profile	0x02*ii*0007	VT_UI2 | VT_VECTOR

Subimage width and height properties (required)
These properties specify the width and height of subimage in pixels. Values do not include the padded area if the image data is padded to tile boundaries.

Subimage color property (required)
This property specifies the color of the subimage channels at this resolution. The format of the data portion of the VT_BLOB is shown in Table 3.4.

TABLE 3.4 Format and fields of the subimage color property

Field name	Length	Byte(s)
Number of subimages	4	0-3
Number of channels of subimage	4	4-7
Color of channel 0 of subimage	4	8-11
…		
Color of channel *last* of subimage	4	variable

Each color code is divided into two sections: the color space and the color. The upper 16 bits of the field specify the color space. The lower 16 bits specify the color.

Valid values for each of the color space subfields are shown in Table 3.5. If the most significant bit of the color space subfield is not set, then the image is calibrated to the color space as defined in *Section 5: Color Space Specifications.* If the most significant bit of the color space subfield is set, then the image channel definitions are that of the color space, but the image is not calibrated. Core reader software should provide a means for warning the application user that the image color is non-standard and unpredictable color results may occur.

TABLE 3.5 Valid color space subfield values

Value	Meaning
0x0	Colorless (CL)
0x1	Monochrome (M)
0x2	PhotoYCC (YCC)
0x3	NIF RGB (NRGB)

Each color space is defined in *Section 5: Color Space Specifications*.

All the channels in the subimage must have the same color space value, including the most significant bit. It is not legal to create the subimage with a PhotoYCC luminance channel and a NIF RGB green channel. Core reader software must verify that the entire subimage color property value is the same for each resolution of the *FlashPix* image object.

If the subimage contains an opacity channel in addition to other color channels, the opacity channel color space codes should be that of the other channels in the subimage. For example, if an opacity channel is placed in the subimage with NIF RGB data, the complete color code for the opacity channel should be 0x00037FFE. If the subimage contains only an opacity channel, the color space should be colorless (0x0000). For example, an opacity channel alone in the subimage should have a complete color code of 0x00007FFE.

Valid values for each of the color subfields are shown in Table 3.6.

TABLE 3.6 **Valid color subfield values**

Color	Value	Allowed color spaces
Monochrome (M)	0x0	M
Red (R)	0x0	NRGB
Green (G)	0x1	NRGB
Blue (B)	0x2	NRGB
PhotoYCC luminance (Y)	0x0	YCC
PhotoYCC chrominance1 (C1)	0x1	YCC
PhotoYCC chrominance2 (C2)	0x2	YCC
Opacity (A)	0x7FFE	all

If the most significant bit of the color subfield (0x8000) is set, that channel is part of a subimage containing opacity data, which, in addition to being included as an additional channel, has been premultiplied into the color channels. If one color channel has been premultiplied, all channels except the opacity channel must be premultiplied. An opacity channel may never have the premultiplied flag set. In color character codes (such as R-NRGB), a lowercase “a” at the beginning of the character code indicates a channel with premultiplied opacity (for example, aR-NRGB). If the primary subimage contains an opacity channel, it must be premultiplied into the other channels.

Table 3.7 shows the valid subimage color value combinations. The channel color definitions are expected to appear in the order in which they are listed under Color codes in the table.

TABLE 3.7 Legal subimage color values

Description	Color codes
PhotoYCC	Y-YCC, C1-YCC, C2-YCC
PhotoYCC with premultiplied opacity	aY-YCC, aC1-YCC, aC2-YCC, A-YCC
NIF RGB	R-NRGB, G-NRGB, B-NRGB
NIF RGB with premultiplied opacity	aR-NRGB, aG-NRGB, aB-NRGB, A-NRGB
Monochrome	M-M
Monochrome with premultiplied opacity	aM-M, A-M
Opacity	A-CL

Subimage numerical format property (required)

This property specifies the numerical formats of the image data at this resolution. The value and the image data may only be of the type VT_UI1 (8-bit unsigned integer). All channels in the subimage must have the same numerical format. Core readers must verify that the same subimage numerical format value is given for all channels of each resolution of each subimage.

Decimation method property (required)

This property characterizes the quality of the decimation performed to create the images at this resolution from the next higher resolution. If the value is 0x7FFFFFFF, this resolution was decimated from the next larger resolution using the following 8-point decimation prefilter:

(-0.046734,-0.059009,0.156544,0.449199,0.449199,0.156544,-0.059009,-0.046734).

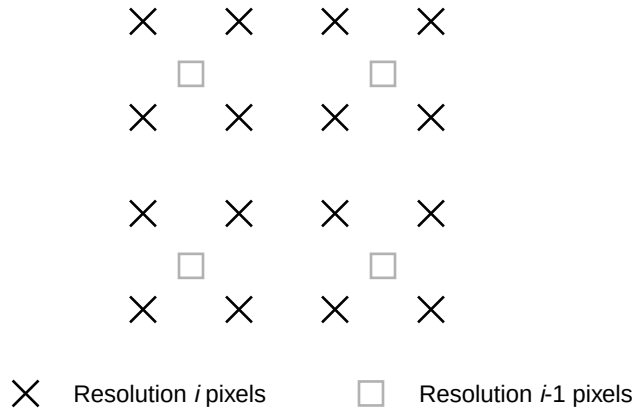
Other values indicate the number of elements in the prefilter. For example, if a 6-point filter was used to prefilter the image before decimation, this value would be 6. If this resolution is the full resolution image, the decimation method should have a value of zero.

If this resolution was artificially created by interpolating from a smaller resolution (it is not recommended for images to be interchanged with other applications), the value of the decimation method property should be negative. The absolute value is the width of the filter applied to sharpen the image prior to interpolating the data. Then -1 indicates a simple interpolation without sharpening.

Correct decimation filter design depends on the choice of pixel location (Figure 3.3). The *FlashPix* format convention requires that the positions of pixels in a resolution layer (except for the full resolution layer) are offset by one half a pixel unit with respect to the resolution layer above. This can be obtained by using an even-width symmetric filter to prefilter the image before decimating. Note that most convolution code places the center of the filter at the location of the filtered pixel and that a nearest neighbor dec-

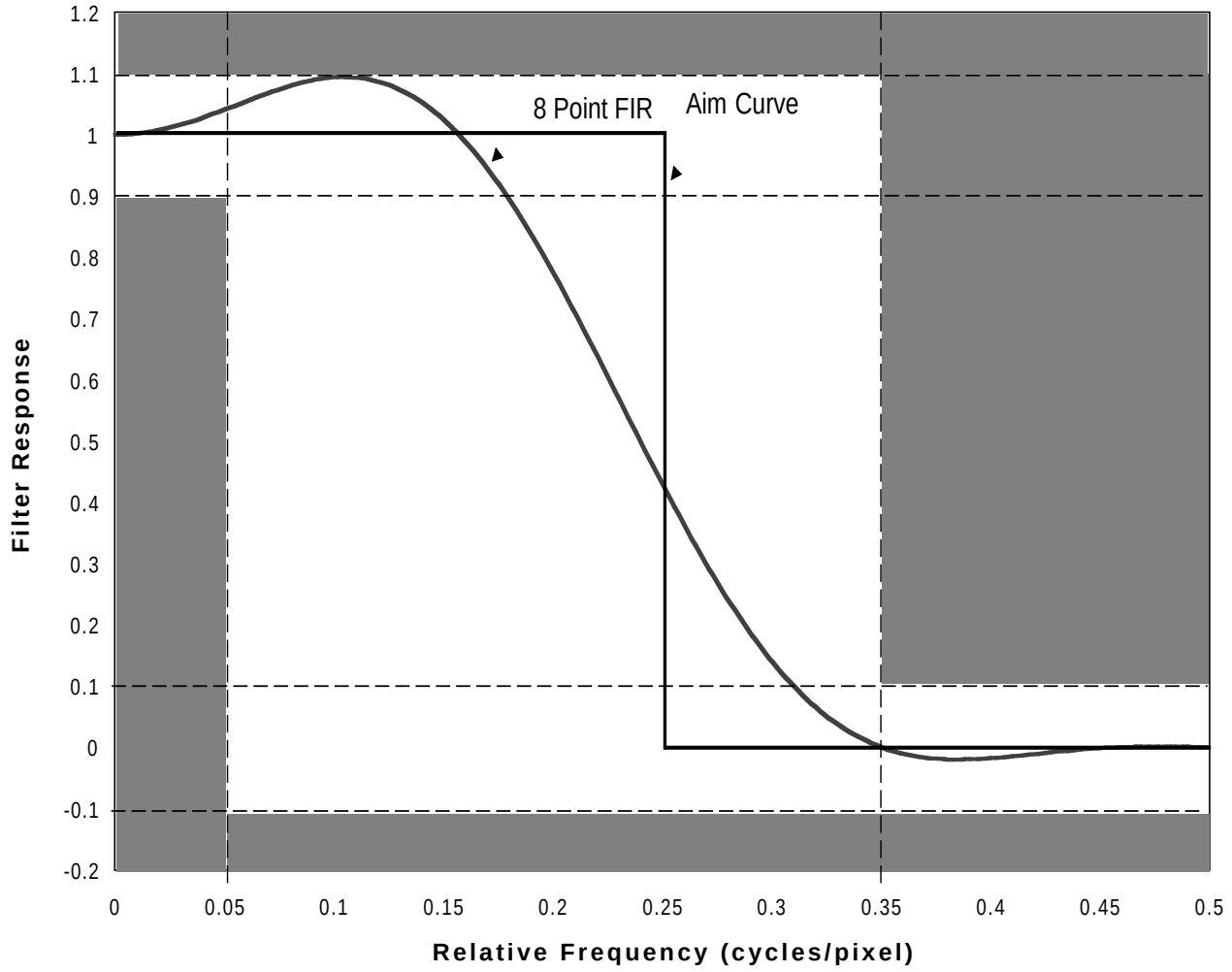
imation (such as the Windows StretchBlt function) produces incorrect pixel alignment and should not be used.

FIGURE 3.3 **Example of pixel-centered alignment between adjacent resolutions**



A developer may choose to design their own decimation prefilter. The filter must be a finite impulse response filter (FIR) and should be designed according to the aim frequency response curve and error bounds shown in Figure 3.4. The filter design process should achieve a prefilter frequency response curve which approximates the aim curve without ever entering the shaded region shown in the figure. A reader should assume that the image was decimated by a filter approximating this frequency response; and a writer should use a filter having the same aim frequency response.

FIGURE 3.4 **Frequency response curve and error bounds**



Decimation prefilter width property (optional)

To perform resolution-independent image filtering, the algorithm must know the decimation prefilter degree of blurring, which is expressed as the effective filter width, *q*. This property specifies the value *q*.

The procedure is as follows: Approximate the prefilter MTF by the form e^{-qs^2}, where *q* is the width and *s* is the spatial frequency measured in cycles per pixel of the image before filtering. If the MTF is far from a Gaussian form, fit the low-frequency portion best. If this property is not found, it is assumed that the prefilter used had an aim frequency response as specified in the decimation method property.

Subimage ICC profile property (optional)
This property specifies an optional ICC profile for the subimage. It is a 1-element array whose value must be 1. The existence of this property indicates that an ICC profile exists in the *FlashPix* image object. The profile specifies the conversion of the subimage into the ICC-PCS (Profile Connection Space). If an ICC profile is identified for the subimage, the same profile must be associated with the subimage at all resolutions and core reader software must verify that this is the case.

3.1.5.3 Compression Description Group

This group of the image contents property set contains compression header information. Only those properties that contain valid data must be present.

Table 3.8 specifies the properties in the compression description group. In the table, "*ii*" in the ID code is the index of a JPEG table set selection.

TABLE 3.8 Valid properties in the compression information properties group

Property name	ID code	Type
JPEG tables	0x03*ii*0001	VT_BLOB
Maximum JPEG table index	0x03000002	VT_UI4

JPEG tables property (optional)
This property (as specified in Table 3.8) contains the JPEG quantization tables and the JPEG Huffman tables used across all resolutions of the *FlashPix* image. The format of the data in each of these properties should conform to the JPEG abbreviated format for table specification data, consisting of JPEG markers surrounding the actual table data, per the JPEG specification[9] Annex B.5 and Annex C. Note that it is possible to use quantizers or Huffman tables not defined here by including them in the JPEG data stream for the tile in which to apply them (as described in Section 4.1.2). The format of a JPEG abbreviated header table is shown in Table 3.9.

TABLE 3.9 Format and entries of a JPEG abbreviated header table

Field name	Length	Byte(s)	Value
Start of image marker (SOI)	2	0-1	0xFFD8
Define quantization table segment marker (DQT)	2	2-3	0xFFDB
Quantization table data	variable	4-variable	variable
Define Huffman table segment marker (DHT)	2	variable	0xFFC4
Huffman table data	variable	variable	variable
End of image marker (EOI)	2	variable	0xFFD9

Each *FlashPix* image tile using JPEG compression must define at least one quantization table and two Huffman tables. A typical JPEG abbreviated table stream includes two quantization tables (numbered 0 and 1) for the luminance and chrominance components, and two Huffman tables (numbered 0 and 1) for the DC and AC entropy coder. The stan-

dard JPEG table specification syntax allows the definition of up to four quantization tables and four Huffman tables.

By storing multiple JPEG abbreviated header tables (each in a uniquely identified property), different sets of tables can be used by different tiles and multiple tiles can utilize the same table. Up to 255 table streams, with indices ranging from 1 to 255, may be defined in the compression property group.

Maximum JPEG table index property (optional)

This field specifies the maximum JPEG table index for the JPEG table properties. This property is optional, but must exist if there are any JPEG table properties in this *FlashPix* image object. It is recommended that when a JPEG table property is added that the index used be this property's value + 1. When a JPEG table property is added, the maximum JPEG table index property must be adjusted if the new index value in use is greater than the current property value.

3.1.6 ICC Profile (optional)

Stream name: ICC**040**Profile**040**0001
Class ID: 56616600-C154-11CE-8553-00AA00A1F95B

This stream contains an ICC profile describing the conversion between the *FlashPix* image color space and the ICC PCS. The data portion of the *FlashPix* stream is stored in standard ICC profile format[8]. The ICC profile may contain only standard PhotoYCC to PCS or NIF RGB to PCS transforms.

3.1.7 Extension List Property Set (optional)

Stream name: **005**Extension**040**List
Class ID: 56616010-C154-11CE-8553-00AA00A1F95B
Format ID: 56616010-C154-11CE-8553-00AA00A1F95B

This property set identifies extensions present in the *FlashPix* image object by class ID, name, and description as well as the data elements changed or added by each extension. The property set is optional, however, if the *FlashPix* image object contains any extensions, the extension list property set must be present and all extension, registered and private, in the *FlashPix* image object must be described.

The way in which the data associated with an extension is structured can take one or more of the following forms:

- New storage(s) may be added
- New stream(s) may be added
- New *FlashPix* stream(s) may be added
- New subimage(s) may be added to a *FlashPix* image object
- New property set(s) may be added
- New property(s) may be added to an existing property set section

- Element(s) may be added to core *FlashPix* property set vector properties that are defined as variable length
- Value of a core *FlashPix* stream field may be changed
- Value of a core property set property may be changed

There are five restrictions to structuring the data elements of an extension. First, new fields may not be added to existing *FlashPix* streams. Second, due to the inability to independently ensure property ID code uniqueness, only registered extensions may add properties to an existing property set section. Third, private extensions may not change the value of a core *FlashPix* stream field or a core property set property. Fourth, extensions can only add vector elements that are not already used by core or other extensions present in the file. Upon removal of an extension, the vector element values associated with the extension must be replaced with NULL and the vector must not be reordered. Fifth, only registered extensions can add elements to core property set vector properties.

Although there are a few practical examples where reasonable core reader actions could be defined for when an extension has changed the value of a core *FlashPix* stream field or a core property set property, these core reader actions must be considered in defining the core *FlashPix* specification. It is impractical to expect all core reader software to be updated to incorporate default actions identified in the course of developing new extensions. The definition of extensions must not impact the core definition unless some compelling feature set is identified which the *FlashPix* format Advisory Council agrees to include in a revised definition of the core *FlashPix* format. Therefore, efforts to define public extensions will avoid impacting core *FlashPix* stream field and core property set property values.

The valid properties of the extension list property set are listed in Table 3.10. The extensions present in the *FlashPix* image object are numbered for the convenience of grouping the descriptive information about each extension. Property ID codes 0xiiiixxxx describe the extension numbered 0x*iiii*.

Used extension numbers property (required)
This property lists all extension numbers iiii used in the extension list property set for the *FlashPix* image object. The property value is an unordered array of iiii values.

All applications must update this property each time an extension is added to or removed from a *FlashPix* image object.

Extension name property (required)
This property identifies the name of the extension. If the extension is registered, the name used must be that which is published in the official*FlashPix* Extension Specification. For private extensions, the name is the whatever short, descriptive label the authoring application chooses.

All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence.

TABLE 3.10 **Valid properties for the extension list property set**

Property name	ID code	Type
Used extension numbers	0x10000000	VT_UI2 | VT_VECTOR
Extension name	0x*iiii*0001	VT_LPWSTR
Extension class ID	0x*iiii*0002	VT_CLSID
Extension persistence	0x*iiii*0003	VT_UI2
Extension creation date	0x*iiii*0004	VT_FILETIME
Extension modification date	0x*iiii*0005	VT_FILETIME
Creating application	0x*iiii*0006	VT_LPWSTR
Extension description	0x*iiii*0007	VT_LPWSTR
Storage / stream pathname	0x*iiii*1000	VT_LPWSTR | VT_VECTOR
FlashPix stream pathname	0x*iiii*2000	VT_LPWSTR | VT_VECTOR
FlashPix stream field offset	0x*iiii*2001	VT_UI4 | VT_VECTOR
Property set pathname	0x*iiii*3000	VT_LPWSTR | VT_VECTOR
Property set ID codes	0x*iiii*3jj1	VT_LPWSTR | VT_VECTOR
Property vector elements	0x*iiii*3jj2	VT_LPWSTR | VT_VECTOR
Subimage number/resolution	0xiiii4000	VT_LPWSTR | VT_VECTOR

Extension class ID property (required)

This property identifies a unique class ID for the extension. If the extension is registered, the class ID must be that which is published in the official*FlashPix* Extension Specification. For private extensions, the class ID is assigned by the authoring application.

All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence.

Extension persistence property (required)

This property identifies the persistence of the extension with respect to edits to the core data elements of the *FlashPix* image object. The legal values for the extension persistence property are defined in Table 3.11.

All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence.

It is the responsibility of the reader/writer application upon save or copy functions to retain the extension data elements by default, or in accordance with the extension persistence property.

TABLE 3.11 **Legal values of the existence persistence property**

Value	Meaning
0x0	Extension is valid independent of core element modifications
0x1	Extension is invalid upon core element modifications
0x2	Extension is potentially invalid upon core element modifications

Extension creation date property (optional unless extension persistence property is 0x2)

This property specifies the time and date the authoring application added the extension to the *FlashPix* image object. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Extension modification date property (optional unless extension persistence property is 0x2)

This property specifies the time and date of the last modification to the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Creating application property (optional)

This property specifies the name of the application that authored the extension in the file. If the property exists, any application editing the extension must update the value and all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Extension description property (optional)

The description property is a short (<80 character) description of the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Storage/stream pathname property (optional)

This property lists the full storage or non-*FlashPix* stream name, including the path in the structured storage file from the *FlashPix* image object storage, for each storage or non-*FlashPix* stream the extension added to the *FlashPix* image object. The path is specified using the standard Unix file specification tokens: "/" represents a directory separator and must be the first character of the property value. Wildcard characters “*” and “?” (where “*” matches any 0 or more characters and “?” matches any 1 character) are permitted in the path portion of the property value. If a storage is listed in the extension list property set, its contents should not also be listed as they are assumed to also be associated with that extension. If this property is omitted it is assumed that no storages are added to the *FlashPix* image object for the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

FlashPix stream pathname property (optional)

This property lists the full *FlashPix* stream name, including the path from the *FlashPix* image object storage, for each *FlashPix* stream the extension added to or modified in the

FlashPix image object. The path is specified using the standard Unix file specification tokens: "/" represents a directory separator and must be the first character of the property value. Wildcard characters “*” and “?” (where “*” matches any 0 or more characters and “?” matches any 1 character) are permitted in the path portion of the property value. The array of values for the *FlashPix* stream pathname property and the *FlashPix* stream field offset property array of values for extension iiii are associated as described in Table 3.12. If this property is omitted it is assumed that no *FlashPix* streams are added to or modified in the *FlashPix* image object for the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

***FlashPix* stream field offset property (optional)**

This property lists the byte offsets (after the header) into the *FlashPix* stream identified with the *FlashPix* stream pathname property array of fields modified by the extension. The array of values for the *FlashPix* stream field offset property and the *FlashPix* stream pathname property array of values for extension 0xiiii are associated as described in Table 3.12. This property is required only if the *FlashPix* stream pathname property exists. If this property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

In the Table 3.12 example, there are two *FlashPix* stream data elements associated with extension 0x17. The first, at Index = 0, is an added *FlashPix* stream as there is a *FlashPix* stream pathname value but the *FlashPix* stream field offset is 0xFFFFFFFF. The second, at index 0xjj = 1, is a field in a core *FlashPix* stream who's value is not among those defined in core the *FlashPix* format. This is indicated by the presence of a non-0xFFFFFFFF *FlashPix* stream field offset value in addition to a *FlashPix* stream pathname value.

TABLE 3.12 **Example values of *FlashPix* stream identification**

Property	Index = 0	Index = 1
0x00172000	stream x pathname	stream y pathname
0x00172001	0xFFFFFFFF	64

Property set pathname property (optional)

This property is an array that lists the full property set name, including the path from the *FlashPix* image object storage, for each property set the extension 0xiiii added, added to, or modified in the *FlashPix* image object. The path is specified using the standard Unix file specification tokens: "/" represents a directory separator and must be the first character of the property value. Wildcard characters "*" and "?" (where "*" matches any 0 or more characters and "?" matches any 1 character) are permitted in the path portion of the property value. Table 3.13 shows an example of how the property set pathname, property set ID codes, and property set vector elements for extension 0xiiii are associated. The array index of the property set pathname property corresponds to 0xjj in the properties 0xiiii3jj1 and 0xiiii3jj2.

This property set is optional and if omitted it is assumed that no property sets are added, added to, or modified in the *FlashPix* image object for the extensions. If the property

exists all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

Property set ID codes property (optional)

This property lists the ID codes of properties which have been added to a core property set, or defined with non-core values by an extension to the *FlashPix* image object. The value of each array posistion of the property is a VT_LPWSTR that may be composed of comma separated values each of which are either an individual property ID code or hyphen-separated pair of property ID codes. The array of values for the property set ID codes and the property vector elements for particular property set 0xjj and extension 0xiiii are associated as described in Table 3.13. When a new property set is added by an extension, the property set ID codes property is not required. This property is required if an extension adds properties to a core property set or modifies core property set properties. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

Property vector elements property (optional)

Extensions can add vector elements to core properties that are defined as variable length vectors. This property lists the vector index for the values added to a particular vector property. The value of each array posistion of the property is a VT_LPWSTR that may be composed of comma separated values each of which are either an individual vector element or hyphen-separated pair of vector elements. The array of values for the property vector elements and the property set ID codes for a particual property set 0xjj and extension 0xiiii are associated as described in Table 3.13. This property is only required when an extension adds vector elements to a core property set property. If the vector elements property is present, and vector elements have not been added to its associated property set ID code(s), then the value of this property must be NULL . If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

In the Table 3.13 example, there are three property sets associated with extension 0x19. The first index of the property 0x000193000, which corresponds to 0xjj=00, is a new property set being added by the extension as there is a property set pathname value, but the property set ID codes and property vector element properties for 0xjj=00 are not listed. The second index of the property 0x00193000, which corresponds to 0xjj=01, is a core property set in which property ID Codes 0x00011001-0x00011005 and 0x00001200 are being added by the extension. The third index of the property 0x00193000, which corresponds to 0xjj=02, is a core property set in which property ID code 0x00033000 is of type VT_VECTOR and the extension has added values in elements 3,4, and 5 of that vector. Property ID codes $00044001-$00044004 are new ID codes being added to the property set by the extension as well. In this case since the property ID codes are new, the value of 0x00193022 for this array posistion is assigned

to NULL. This example also shows that the extension has added a value in element 2 of both the vectors defined by existing property ID codes, 0x00055000 and 0x00066000.

TABLE 3.13 **Example values of property set identification**

Property	Index=0	Index=1	Index=2
000193000	PS x pathname (0xjj=00)	PS y pathname (0xjj=01)	PS z pathname (0xjj=02)
000193011	$000011001-$000011005, $000012000		
000193021	$000033000	$000044001-$000044004	$000055000, $000066000
000193022	3,4,5	NULL	2

Subimage number/resolution property (optional)

This property is used to indicate that an extension has added new subimages to a *FlashPix* image object. It must be not used to indicate that an extension has modified core *FlashPix* subimage 0. The value of each array position is a VT_LPWSTR that is composed of comma-separate values each of which is either an individual subimage number or a hyphen-separated pair of subimage numbers. The array element position indicates the resolution indices to which the particular subimage(s) are added. If subimages are not added to a particular resolution, the value of the corresponding array element must be set to NULL.

The extension list must also list any additional storages, streams, property sets and properties added by the extension to store the subimage. This property is required if an extension adds a subimage to a *FlashPix* image object. All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence. Table 3.14 is an example of an extension that adds subimages to the *FlashPix* image object.This example shows that extension 0x49 has added subimages to resolution 0, 1, and 2 of the core *FlashPix* image object. The first element, at index 0xjj=0, shows that subimages 1 and 3 have been added to resolution 0x00. No subimages have been added to resolution 0x01. The third element, at index 0xjj=2, shows that the extension has added subimages 1, 3, and 4 through 6 to resolution 0x02.

TABLE 3.14 **Example of subimage identification**

Property	0xjj=0	0xjj=1	0xjj=2
0x00494000	1,3	NULL	1,3,4-6

SECTION 4

Image Data Format

The subimage is stored as a separate entity within the *FlashPix* image object. This chapter specifies the format of the subimage, as described in Section 3.1.1.

The subimage is stored in two *FlashPix* streams: a header stream and a data stream.

4.1 The Subimage Header Stream

Stream name: Subimage**040**0000**040**Header
Class ID: 00010000-C154-11CE-8553-00AA00A1F95B

The subimage header stream determines image data location in the data stream and contains information for decoding the data stream into uncompressed pixel values. Values are stored in little endian byte order.

4.1.1 Subimage Header Stream Data

The format of the data portion of the subimage header stream is given in Table 4.1.

TABLE 4.1 **Format and fields of the subimage header stream**

Field name	Length	Byte(s)
Length of header stream header	4	0-3
Image width	4	4-7
Image height	4	8-11
Number of tiles	4	12-15
Tile width	4	16-19
Tile height	4	20-23
Number of channels	4	24-27
Offset to tile header table	4	28-31
Length of tile header entry	4	32-35
Tile header table	variable	variable

Length of header stream header field

This field specifies the length of the header stream header (from the beginning of the length field to the end of the length of tile header entry field) in bytes.

Image width and height field

These fields specify the width and height of the subimage in pixels. These values do not include any padding at the right or bottom of the image to fill partial tiles. The values of these fields must be the same as the values of the subimage width and height in the primary description group of the image contents stream (Section 3.1.5.1, Table 3.1).

Number of tiles field

This field specifies the number of tiles in the subimage.

Tile width field

This field specifies the width of a tile in pixels. The tile width must be 64 pixels.

Tile height field

This field specifies the height of a tile in pixels. The tile height must be 64 pixels.

Number of channels field

This field specifies the number of channels in the subimage. This value is also specified in the image contents property set (Section 3.1.5, Table 3.4).

Offset to tile header table field

This field specifies the offset in bytes from the beginning of the data portion of the *FlashPix* stream to the tile header table (Section 4.1.2).

Length of a tile header entry field

This field specifies the length of a single entry in the tile header table (Section 4.1.2).

Tile header table field

This field specifies the header data for each tile. The format of a tile header table is specified in Section 4.1.2.

4.1.2 Tile header table

The format of the tile header table is given in Table 4.2.

TABLE 4.2 **Format and fields in the tile header table**

Field Name	Length	Byte(s)
Tile header 0	16	0-15
…		
Tile header *last*	16	variable

Tile header 0-*last*

These fields specify the location and encoded form of the image data tiles. Tiles are ordered from top to bottom, left to right, in row major order. The tile containing pixel (0, 0) is first, followed by the tile containing pixel (*tile width*, 0). This order continues across the row, through the tile containing pixel (*width* - *tile width*, 0). Subsequent rows of tiles follow in the same order. Table 4.3 specifies the format of a tile header.

TABLE 4.3 **Format and fields of a tile header**

Field name	Length	Byte(s)
Tile offset	4	0-3
Tile size	4	4-7
Compression type	4	8-11
Compression subtype	4	12-15

Tile offset field

This field specifies the offset of the tile data from the beginning of data portion of the subimage data *FlashPix* stream in bytes. This value is zero if the compression algorithm requires no data other than the compression type and compression subtype fields.

Tile size field

This field specifies the size of the tile data for this tile, in bytes. This value is zero if the compression algorithm requires no data other than the compression type and compression subtype fields.

Compression type field

This field specifies the compression algorithm used to encode the data for this tile. Valid compression type values are given in Table 4.4.

TABLE 4.4 **Valid compression type values**

Value	Meaning
0x0	Uncompressed data
0x1	Single color compression (4-byte)
0x2	JPEG (8-bit)
0xFFFFFFFF	Invalid tile

The invalid tile compression type may be used to temporarily indicate that the tile has no valid data. Situations where marking a tile as invalid may be useful are during resolution hierarchy regeneration or during a partial resynchronize operation between resolutions. Images with invalid tiles should not be saved permanently.

Images with any tile having the invalid tile compression type are considered to be invalid. If reader software encounters the invalid tile compression type when preparing to access a tile, it has no responsibility to attempt to create usable image data from higher resolutions. It is permitted to respond as though a read error occurred.

Compression subtype field

This field specifies compression algorithm information for this tile. The format of this field depends on the value of the compression type field. The different formats are described below.

If the compression type is set to uncompressed data (0x0) or invalid tile (0xFFFFFFFF), the compression subtype is unused and must be set to 0x0.

If the compression type is set to single color compression (0x1), the compression subtype identifies the actual color value of all pixels in the tile. Individual channel values are stored in little endian format, in the same order and bit depth as specified by the subimage color and subimage numerical format properties (Section 3.1.5.3), aligned at the 0th bit of the field.

If the compression type is set to JPEG compression (0x2), the compression subtype field

TABLE 4.5 **Format and entries of the compression subtype field for JPEG compressed tiles**

Field name	Length	Byte(s)
Interleave type	1	0
Chroma subsampling	1	1
Internal color conversion	1	2
JPEG tables selector	1	3

will contain additional information needed by the reader to process the JPEG compressed data. The format of the compression subtype subfields is shown in Table 4.5.

Interleave type subfield

This field specifies the interleaving of the data within the JPEG data stream. If the value is 0x0, all channels in the tile are stored in a single scan with the 8×8 blocks for each channel interleaved. If the value is 0x1, each channel is stored as a separate scan. All other values are illegal. In either case, the channels are found in the same order as specified by the subimage color property of the image contents property set.

Chroma subsampling subfield

This field specifies the amount of subsampling performed on chroma components of the image (either the native components of some YCrCb image or those generated by rotating some RGB image through the standard JPEG CCIR 601 RGB to YCrCb conversion). The most significant nibble of the field indicates the horizontal subsampling ratio. The least significant nibble of the field indicates the vertical subsampling ratio. Legal values of the subsampling fields are 1 and 2.

Both the values for horizontal and vertical subsampling must be either 1 or 2, and if horizontal subsampling is 1, then vertical subsampling must also be 1. The specific horizontal and vertical subsampling pairs (h.v) allowed are (2,2), (2,1), and (1,1). Subsampling in the horizontal direction by 2x and the vertical direction by 1x is allowed for compatibility with digital video standards.

Under no circumstances should an Opacity channel be subsampled.

Internal color conversion subfield

This field specifies whether a color conversion was performed in the JPEG compression process. Valid values are 0x0 and 0x1. All other values are illegal.

If the value is 0x0, no color conversion was performed and the pixel values output from the JPEG decoder are in the color space specified by the subimage color value from the Image Contents property set. If the field value is 0x1, the effect of this field is dependent on the existence of an Opacity channel as described below.

For NIF RGB subimage color value:

If the color space specified by the subimage color value (Table 3.7) is NIF RGB (whether calibrated or not calibrated), i.e. NIF RGB with no opacity channel, then the following standard RGB to YCrCb conversion (or an equivalent integer implementation) is performed on the input data in the JPEG encoding process:

$$Y = 0.29900*R + 0.58700*G + 0.11400*B \quad (4.1)$$

$$Cb = (B - Y)/1.772 + 0.5 = -0.16874*R - 0.33126*G + 0.50000*B + 0.5 \quad (4.2)$$

$$Cr = (R - Y)/1.402 + 0.5 = 0.50000*R - 0.41869*G - 0.08131*B + 0.5 \quad (4.3)$$

where R, G, B, Y, Cb and Cr values are in the 0 ... 1 range.

When decoding, the inverse transformation from YCrCb to RGB is done according to the following equations (or an equivalent integer implementation):

$$R = Y + 1.40200*Cr - 0.70100 \quad (4.4)$$

$$G = Y - 0.34414*Cb - 0.71414*Cr + 0.52914 \quad (4.5)$$

$$B = Y + 1.77200*Cb - 0.88600 \quad (4.6)$$

For NIF RGB with premultiplied opacity subimage color value:

If the value is 0x1 and the color space specified by the subimage color value is NIF RGB with premultiplied opacity (whether calibrated or not calibrated), then to retain interoperability with early *FlashPix* applications, the RGB input was 'inverted' before the standard RGB to YCbCr transform was applied. Please note that the opacity channel is not affected by this operation and must not be inverted or color converted. The sequence of conversion steps is:

(a) Invert the RGB values, i.e., for RGB encoded with 8 bits calculate new color values R' = (255-R), G'= (255-G) and B'= (255-B).

(b) Transform R'G'B' to the new space Y'Cb'Cr' using equations (4.1), (4.2), (4.3).

Compression is done in the Y'Cb'Cr' space. On the decoder side, the inverse transformation should take place, i.e.,

(c) Transform Y'Cb'Cr' to R'G'B' using equations (4.4), (4.5), (4.6).

(d) Transform R'G'B' to RGB, e.g., R = (255-R').

Please note that the current requirement for this legacy 'inversion' results in minor differences when compared to compression done to RGB without opacity channels, but may preclude the use of some kinds of hardware acceleration.

JPEG tables selector subfield

This byte selects a set of quantizer and Huffman tables to use to decompress this tile. If the index is 0x0, the tables are included at the beginning of the tile data stream and it is not necessary to load a separate tile stream into the JPEG decompressor to decompress the tile. A value from 1 to 255 indicates that this tile uses a set of tables stored in one of the JPEG tables properties in the compression property group (Section 3.1.5.3). Specifically, if the value is 0x*ii*, a *FlashPix* reader should load the value of property 0x03ii0001 into the JPEG decompressor.

4.2 The Subimage Data Stream

Stream name: Subimage\0400000**040**Data
Class ID: 00010100-C154-11CE-8553-00AA00A1F95B

The subimage data stream contains the data referenced by the tile headers in the header stream.

4.2.1 Channel Ordering

The channels of multi-channel tiles are ordered as specified in the color space property in the image contents property set (Section 3.1.5.2).

4.2.2 Tile Data Format

4.2.2.1 Uncompressed

Data in uncompressed tiles is stored in row major order in a pixel interleaved fashion. Pixel channels are ordered as specified by the color space property in the image contents property set. Pixel values are stored in little endian format in the type specified by the numerical format property.

4.2.2.2 Single Color Compressed

No tile data is needed for single-color compressed tiles. Both the tile data offset and tile size in the tile header table must be set to zero.

4.2.2.3 JPEG Compressed

The format of compressed tile data conforms to the “Abbreviated Format for Compressed Image Data” described in Annex B, Section B.4 of the ISO JPEG Specifications [9]. At a minimum, this format contains the following JPEG markers and marker segments, as well as the entropy-coded data for the tile (Table 4.6).

Quantizer tables and Huffman table marker segments are not required, but may be included to force a decoder to use tables other than those defined in Section 3.1.5.3.

TABLE 4.6 **Format and entries of a JPEG abbreviated format stream for tile data**

Field name	Length	Byte(s)	Value
Start of image marker (SOI)	2	0-1	0xFFD8
Start of frame marker (SOF)	2	2-3	0xFFC0
Frame header	variable	4-variable	variable
Start of scan marker (SOS) 0	2	variable	0xFFDA
Scan header 0	variable	variable	variable
Entropy coded data 0	variable	variable	variable
…			
Start of scan marker (SOS) *last*	2	variable	0xFFDA
Scan header *last*	variable	variable	variable
Entropy coded data *last*	variable	variable	variable
End of image marker (EOI)	2	variable	0xFFD9

Note that some JPEG CODECS may identify the encoded color space through JPEG specific markers. The *FlashPix* format provides other mechanisms for identifying color in the Image Contents property set which are the subimage color property and the color space subfield value. No other mechanism or values, besides those found in the Image Contents property set, can be utilized to make any decisions about what color space is intended for a given *FlashPix* file.

SECTION 5

Color Space Specifications

5.1 Introduction

The method of encoding for color imagery is critical to how consistently the colors in an image will be reproduced across different systems and different media types. The *FlashPix* format defines two colorspaces, PhotoYCC and NIF RGB, along with respective reference viewing environments for the flexible and unambiguous encoding. The *FlashPix* format also provides a well-defined monochrome encoding space for the storage of greyscale imagery and optional support for InterColor Consortium(ICC) color management used in conjunction with the *FlashPix* color encoding.

PhotoYCC and NIF RGB color values represent color appearance with respect to a defined reference viewing environment. For color stimuli that are meant to be viewed in the reference viewing environment, PhotoYCC and NIF RGB values are computed by a series of simple mathematical operations from standard CIE colorimetric values. For color stimuli that are meant to be viewed in an actual viewing environment that is different from the reference environment, it is necessary to include appropriate colorimetric transformations to determine visually corresponding CIE colorimetric values for the reference environment (the corresponding CIE colorimetric values define a stimulus that, if viewed in the reference viewing environment, would produce the same color appearance as the actual stimulus viewed in the actual environment). These transformations account for differences in the amount of viewing flare in the actual and reference environments, as well as for alterations in observer perception that would be induced by the differences in the environments. The corresponding CIE colorimetric values resulting from these transformations are then encoded in terms of PhotoYCC or encoded in terms of NIF RGB.

5.2 PhotoYCC and NIF RGB Reference Viewing Environments

Reference viewing environments are defined for both PhotoYCC and NIF RGB in Table 5.1. The reference viewing environments are provided to give a single aim for each color space to allow for the unambiguous definition of color for use in interchange. The two definitions serve different purposes, yet with proper colorimetric and color appearance transformations, it is possible to encode values in one space which were originally encoded in the other.

TABLE 5.1 **Comparison of PhotoYCC and NIF RGB viewing environments**

Condition	PhotoYCC	NIF RGB
Viewing flare	None	1.0%
Image surround	Average	20%
Illuminance level/Luminance level	> 5,000 lux	80 cd/m^2
Adaptive white	$x = 0.3127$, $y = 0.3290$	$x = 0.3127$, $y = 0.3290$

5.2.1 PhotoYCC Reference Viewing Environment

The PhotoYCC reference viewing environment corresponds to conditions typical of outdoor scenes.

- *Viewing flare* is specified as “none.” Any flare light in an original-scene environment is part of the scene itself.
- The *image surround* is defined as “average.” Scene objects typically are surrounded by other similarly illuminated objects.
- The *illuminance level* is representative of typical daylight levels. Note that the illuminance is at least an order of magnitude higher than average indoor levels.
- The chromaticities of the *adaptive white* are those of CIE D65. An adaptive white is defined here as a color stimulus that an observer would judge as perfectly achromatic, with a luminance corresponding to that of a perfect white diffuser. While the chromaticities of the adaptive white will most often be those of the scene illuminant, they may, in certain cases, also be quite different. For example, the observer may be only partially adapted to the illuminant. The adaptive white therefore defines only the chromatic adaptive state of the observer. The adaptive white does *not* define the chromaticities or the spectral power distribution of the scene illuminant.

5.2.2 NIF RGB Reference Viewing Environment

The NIF RGB reference viewing environment corresponds to conditions typical of indoor viewing of computer CRT monitors.

- *Viewing flare* is specified to be 1.0% of the maximum white-luminance level.
- The *image surround* is defined as “20%” of the maximum white luminance. This is close to a CIELAB L* value of 50, while maintaining computational simplicity. The areas surrounding the image being viewed are similar in luminance and chrominance to the image itself. This surround condition would correspond, for example, to an image displayed on a computer monitor where the image on the CRT screen is surrounded by a gray background equivalent to twenty percent of the maximum white.
- The *luminance level* is representative of typical CRT display levels. Note that the illuminance is at least an order of magnitude lower than average outdoor levels.
- The chromaticities of the *adaptive white* are those of CIE D65. An adaptive white is defined here as a color stimulus that an observer would judge as perfectly achromatic, with a luminance corresponding to that of a perfect white diffuser. While the chromaticities of the adaptive white will most often be those of the viewing illuminant, they may also, in certain cases, be quite different. For example, the observer may be only partially adapted to the illuminant. The adaptive white therefore defines only the chromatic adaptive state of the observer. It does *not* define the chromaticities or the spectral power distribution of the viewing illuminant.

5.3 Colorimetric Definitions and Digital Encodings

PhotoYCC and NIF RGB in combination with their reference viewing environments can be defined from standard CIE colorimetric values through simple mathematical transformations. Resulting colorimetric values can then be encoded in terms of digital code values for storage in a *FlashPix* Image.

While NIF RGB and PhotoYCC encode colors using similar standards and equations, the definitions presented in this specification do not constitute a means of conversion between the two color spaces. The conversions are an implementation topic.

5.3.1 PhotoYCC Colorimetric Definition and Digital Encoding

PhotoYCC is defined from standard CIE colorimetric values and the PhotoYCC reference viewing environment which corresponds to daylight-illuminated outdoor scenes. This definition describes the encoding of a daylight-illuminated scene, captured using a Photo CD Reference Image-Capture Device, when the observer adaptive white corresponds to D65 chromaticities. Examples of the encoding of colors not represented by this definition are given in the *FlashPix* Implementation Guide.

The red, green, and blue spectral responsivities of the Reference Image-Capture Device in Figure 5.1 correspond to the color-matching functions for the reference primaries defined in CCIR Recommendation 709[1]. The CIE chromaticities for the red, green, and blue CCIR-709 reference primaries, and for CIE Standard Illuminant D65, are given in Table 5.2.

FIGURE 5.1 **Spectral responsivities of the Reference Image-Capture Device**

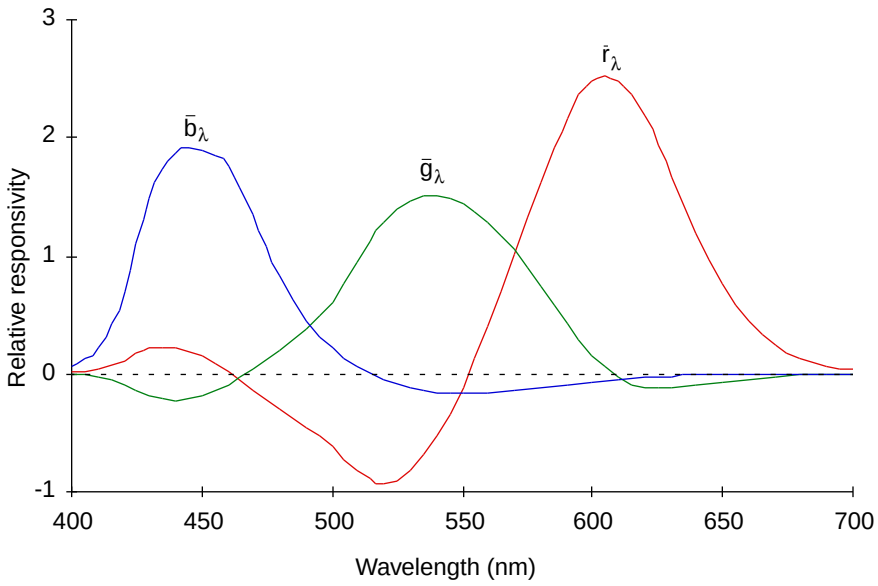


TABLE 5.2 **CIE chromaticities for CCIR-709 reference primaries and CIE standard illuminant**

	Red	Green	Blue	D65
x	0.6400	0.3000	0.1500	0.3127
y	0.3300	0.6000	0.0600	0.3290
z	0.0300	0.1000	0.7900	0.3583
u'	0.4507	0.1250	0.1754	0.1978
v'	0.5229	0.5625	0.1579	0.4683

Reference Image-Capture Device RGB_{709} tristimulus values for the illuminated objects of the scene can be calculated using the spectral responsivities of the Reference Image-Capture Device:

$$\begin{aligned} R_{709} &= k_r \sum_{\lambda} P_\lambda R_\lambda \bar{r}_\lambda \\ G_{709} &= k_g \sum_{\lambda} P_\lambda R_\lambda \bar{g}_\lambda \\ B_{709} &= k_b \sum_{\lambda} P_\lambda R_\lambda \bar{b}_\lambda \end{aligned} \tag{5.1}$$

where P_λ is the spectral power of the scene illuminant at each wavelength λ; R_λ is the spectral reflectance or transmittance of a scene object; and $\bar{r}_\lambda$, $\bar{g}_\lambda$, and $\bar{b}_\lambda$ are the spectral responsivities of the Reference Image-Capture Device. Normalizing factors k_r, k_g, and k_b are determined such that *R*, *G*, and *B* tristimulus values of 1.00 will result for a perfect white diffuser. It is assumed that the reference image capture device produces flareless measurements; it is therefore unnecessary to adjust the resulting RGB values for instrument flare.

Since the spectral responsivities of the Reference Image-Capture Device are simply linear combinations of the 1931 CIE color-matching functions, $\bar{x}_\lambda$, $\bar{y}_\lambda$ and $\bar{z}_\lambda$ [3] its *RGB* tristimulus values can also be computed using the following relationship:

$$\begin{bmatrix} R_{709} \\ G_{709} \\ B_{709} \end{bmatrix} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X_{scene} \\ Y_{scene} \\ Z_{scene} \end{bmatrix} \tag{5.2}$$

where

$$\begin{aligned} X_{scene} &= k_r \sum_{\lambda} P_\lambda R_\lambda \bar{x}_\lambda \\ Y_{scene} &= k_g \sum_{\lambda} P_\lambda R_\lambda \bar{y}_\lambda \\ Z_{scene} &= k_b \sum_{\lambda} P_\lambda R_\lambda \bar{z}_\lambda \end{aligned} \tag{5.3}$$

In the PhotoYCC encoding process, negative RGB_{709} tristimulus values, and RGB_{709} tristimulus values greater than 1.00 are retained. The luminance dynamic range and the color gamut defined by the RGB tristimulus values of the Reference Image-Capture Device are therefore unlimited.

Reference Image-Capture Device RGB_{709} tristimulus values are next transformed to nonlinear $R'G'B'_{709}$ values as follows:

For $R_{709}, G_{709}, B_{709} \geq 0.018$:

$$\begin{aligned} R'_{709} &= 1.099 \times R_{709}^{0.45} - 0.099 \\ G'_{709} &= 1.099 \times G_{709}^{0.45} - 0.099 \\ B'_{709} &= 1.099 \times B_{709}^{0.45} - 0.099 \end{aligned} \quad (5.4a)$$

For $R_{709}, G_{709}, B_{709} \leq -0.018$:

$$\begin{aligned} R'_{709} &= -1.099 \times |R_{709}|^{0.45} + 0.099 \\ G'_{709} &= -1.099 \times |G_{709}|^{0.45} + 0.099 \\ B'_{709} &= -1.099 \times |B_{709}|^{0.45} + 0.099 \end{aligned} \quad (5.4b)$$

For $-0.018 < R_{709}, G_{709}, B_{709} < 0.018$:

$$\begin{aligned} R'_{709} &= 4.50 \times R_{709} \\ G'_{709} &= 4.50 \times G_{709} \\ B'_{709} &= 4.50 \times B_{709} \end{aligned} \quad (5.4c)$$

For PhotoYCC, the nonlinear $R'G'B'_{709}$ values are rotated to luma and chromas as in Equation 5.5:

$$\begin{bmatrix} \text{Luma} \\ \text{Chroma}_1 \\ \text{Chroma}_2 \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.299 & -0.587 & 0.886 \\ 0.701 & -0.587 & -0.114 \end{bmatrix} \begin{bmatrix} R'_{709} \\ G'_{709} \\ B'_{709} \end{bmatrix} \quad (5.5)$$

For PhotoYCC, luma/chroma signals are converted to digital values. For 24-bit (8-bits/channel) encoding, PhotoYCC values are formed according to Equation 5.6:

$$\begin{aligned} Y &= (255/1.402) \times \text{Luma} \\ C_1 &= 111.40 \times \text{Chroma}_1 + 156 \\ C_2 &= 135.64 \times \text{Chroma}_2 + 137 \end{aligned} \quad (5.6)$$

5.3.2 NIFRGB Colorimetric Definition and Digital Encoding

NIFRGB is defined from standard CIE colorimetric values and the NIFRGB reference viewing environment which corresponds to indoor viewing of computer CRT displays. This definition describes the encoding of the appearance of the colors displayed on a reference monitor based on the reference primaries and transfer function implied in

CCIR Recommendation 709[1] when the observer adaptive white corresponds to D65 chromaticities. This transfer function is consistent with a large variety of legacy images including video and Microsoft Windows based imagery.

The CIE chromaticities for the red, green, and blue CCIR-709 reference primaries, and for CIE Standard Illuminant D65[2], are given in Table 5.2.

For NIFRGB, the goal is to communicate the appearance of the presentation of the appearance of the colors as displayed on a reference monitor in terms of 8-bit digital code values. Given the CIE XYZ_{D65} tristimulus values for the colors represented on the monitor, a transformation can be made to reference monitor RGB_{NIF} tristimulus values.

$$\begin{bmatrix} R_{NIF} \\ G_{NIF} \\ B_{NIF} \end{bmatrix} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X_{D65} \\ Y_{D65} \\ Z_{D65} \end{bmatrix} \tag{5.7}$$

In the NIFRGB encoding process, RGB_{NIF} values between 0.0 and 1.0 are encoded, while values outside that range are clipped and not retained. Therefore the luminance dynamic range and color gamut defined by the RGB tristimulus values of the reference monitor are limited.

Reference monitor RGB_{NIF} tristimulus values are next transformed to nonlinear RGB'_{NIF} values as follows:

For $R_{NIF}, G_{NIF}, B_{NIF} \geq 0.00304$:

$$\begin{aligned} R'_{NIF} &= 1.055 \times R_{NIF}^{\,0.42} - 0.055 \\ G'_{NIF} &= 1.055 \times G_{NIF}^{\,0.42} - 0.055 \\ B'_{NIF} &= 1.055 \times B_{NIF}^{\,0.42} - 0.055 \end{aligned} \tag{5.8a}$$

For $0.0 < R_{709}, G_{709}, B_{709} < 0.00304$:

$$\begin{aligned} R'_{NIF} &= 12.92 \times R_{NIF} \\ G'_{NIF} &= 12.92 \times G_{NIF} \\ B'_{NIF} &= 12.92 \times B_{NIF} \end{aligned} \tag{5.8b}$$

For NIF RGB, the nonlinear $R'G'B'_{NIF}$ values are converted to digital code values. For 24-bit (8-bits/channel) encoding, NIF RGB values are formed according to the following Equation 5.9:

$$\begin{aligned} R_{8bit} &= 255.0 \times R'_{NIF} \\ G_{8bit} &= 255.0 \times G'_{NIF} \\ B_{8bit} &= 255.0 \times B'_{NIF} \end{aligned} \tag{5.9}$$

5.4 Monochrome Encoding Definition

The *FlashPix* format supports the encoding and storage of 8-bit monochrome imagery. NIF monochrome is defined in terms of luminance, $Luma_{NIF}$, and is the defined in terms of luminance, $Luma_{NIF}$, and is designed to encode the appearance of a monochrome image on a reference monitor based on the primaries and tone transfer function defined in CCIR Recommendation 709.

Section 2.1 of CCIR Recommendation 601-4 defines a relationship between an analog luminance signal, E'_y and red, green, and blue analog color signals (E'_R, E'_G, E'_B) and that relationship is given here as Equation 5.10.

$$E'_y = 0.299E'_R + 0.587E'_G + 0.114E'_B \tag{5.10}$$

The definition of NIF monochrome uses this relationship, however, the definitions and values of E'_y, E'_R, E'_G, E'_B are not used from CCIR Recommendation 601-4. Instead, NIF monochrome is defined in terms of a luminance, $Luma_{NIF}$, and builds from the non-linear, R'_{NIF}, G'_{NIF}, and B'_{NIF} signals given in Equation 5.8a and Equation 5.8b of the definitions of NIFRGB. The relationship is given in Equation 5.11.

$$Luma_{NIF} = 0.299R'_{NIF} + 0.587G'_{NIF} + 0.114B'_{NIF} \tag{5.11}$$

The 8-bit digital encoding of *Luma* for NIF Monochrome is given in Equation 5.12.

$$Y_{NIF} = 255.0 \times Luma_{NIF} \tag{5.12}$$

SECTION 6

Image Info Property Set

In addition to image data, a *FlashPix* image object also contains information to enhance the use of the image (information, for example, about the image itself, as well as how it was captured and how it might be used). This non-image data is stored in the image info property set in the *FlashPix* image object storage.

Stream name:	**\005**Image**040**Info
Class ID:	56616500-C154-11CE-8553-00AA00A1F95B
Format ID:	56616500-C154-11CE-8553-00AA00A1F95B

6.1 Informational Groups

Though the properties may appear in any order, the property set is divided into several conceptual groups, each describing a different aspect of the image. The property groups are:

- File source
- Intellectual property
- Content description
- Camera information
- Per picture camera settings
- Digital camera characterization
- Film description
- Original document scan description
- Scan device

The information in these groups provides the framework to document facts about image capture, intellectual property concerns, and descriptive information about the image itself. With some images, users need to know who is in the picture, where and when it was taken, and so on, to understand the significance of the image.

For instance, a photograph of an automobile accident is useless to an insurance company unless it is known to which accident the picture applies. Similarly, an old family picture is far more interesting if it is known which ancestor is in the picture, and when and where it was taken. One problem with traditional methods of dealing with images is that it is easy for this data to become separated from the images, greatly diminishing the value of the images.

A fundamental concept of the *FlashPix* format is that an image should be as self-describing as possible. As an image moves across a network, or is written to various types of media, the self-describing data should move with the image.

Any property may be omitted. If omitted, that property should be treated as if the value is unknown. All property ID codes not explicitly listed are reserved for registered extensions.

6.2 File Source Group

This group of properties specify how the image was created. Table 6.1 lists the properties in this group:

TABLE 6.1 **Properties in the file source group**

Property name	ID code	Type
File source	0x21000000	VT_UI4
Scene type	0x21000001	VT_UI4
Creation path vector	0x21000002	VT_UI4 | VT_VECTOR
Software Name/Manufacturer/Release	0x21000003	VT_LPWSTR
User defined ID	0x21000004	VT_LPWSTR
Sharpness approximation	0x21000005	VT_R4

File source property (optional)
This property specifies the device source of the digital file, such as a film scanner, reflection print scanner, or digital camera. Possible values are listed in Table 6.2. Values greater than 0x5 must be handled by core reader software as though they were Unidentified (0x0).

TABLE 6.2 **Valid file source property values**

Value	Meaning
0x0	Unidentified
0x1	Film scanner
0x2	Reflection print scanner
0x3	Digital camera
0x4	Still from video
0x5	Computer graphics

Scene type property (optional)

This property specifies the type of scene that was captured. It differentiates “original scenes” (direct capture of real-world scenes) from “second generation scenes” (images captured from pre-existing hardcopy images). It provides further differentiation for scenes that are digitally composed. Values greater than 0x3must be handled by core reader software as though they were Unidentified (0x0). Possible values are listed in Table 6.3.

TABLE 6.3 **Valid scene type property values**

Value	Meaning
0x0	Unidentified
0x1	Original scene
0x2	Second generation scene
0x3	Digital scene generation

Creation path vector property (optional)

This property encodes the conversion path that an image takes as defined by both analog and digital capture processes. Each element of the vector is a property ID that corresponds to a property in the non-image data *FlashPix* properties. The first element of the vector is the *last* capture of the scene, corresponding to the property for File Source. Some examples are as follows.

A reflection print from an original scene would be listed as “File source, Type of reflection original, Film type, Camera model name, Scene type.”

A reflection print from a second generation scene would be listed as “File source, Type of reflection original, Film type, Camera model name, Original medium, Scene type.”

Film from the original scene would be listed as “File source, Film type, Camera model name, Scene type.”

Film from a second generation scene would be listed as “File source, Film type, Camera model name, Original medium, Scene type.”

An image from a digital camera capture of the original scene would be listed as “File source, Camera model name, Scene type.”

An image from a digital camera capture of a second generation scene would be listed as “File source, Camera model name, Original medium, Scene type.”

A still from a video camera would be listed as “File source, Camera model name, Scene type.”

A computer generated image would be listed as “File source, Software name/manufacturer/release.”

Software name/release property (optional)

This property encodes the name of the software, its manufacturer’s name, and the version of the software used to create the *FlashPix* image.

User defined ID property (optional)

This property encodes the values of an identification system assigned to an image by the user. This property is useful when users have their *own* filing or accounting scheme with an identification system already in place, and enables users to cross-reference their digital files to a pre-existing analog one.

Sharpness approximation property (optional)

To perform image filtering in a resolution independent manner (Section 7.3.2), the algorithm must have information on the degree of blurring introduced by the system components which generated the digital image (digital camera, scanner, etc.). This is expressed as the effective filter width, q. Approximate the total capture MTF by the form e^{-qs^2}, where q is the width and s is the spatial frequency measured in cycles per pixel at the captured resolution delivered by the input device. If the MTF is far from Gaussian form, fit the low-frequency portion best. This property specifies the value q.

6.3 Intellectual Property Group

The intellectual property group contains information about the ownership and copyright status of the image. Rights for an original artifact may be stated, along with the rights for the digital file. Table 6.4 lists the properties in this group.

Copyright message property (optional)

This property encodes the copyright notice of the Legal Broker for the digital file. The complete copyright statement should be listed in this field, including any dates and statements of claims. If desired, this property can also list details concerning the Legal Broker.

TABLE 6.4 **Properties in the intellectual property group**

Property name	ID code	Type
Copyright message	0x22000000	VT_LPWSTR
Legal broker for the original image	0x22000001	VT_LPWSTR
Legal broker for the digital image	0x22000002	VT_LPWSTR
Authorship	0x22000003	VT_LPWSTR
Intellectual property notes	0x22000004	VT_LPWSTR

Legal broker for the original image property (optional)

This property encodes the name of the person or organization that holds the legal right to grant permissions or restrict use of the original image. The original image is either the analog source scanned to create the digital file or the original digital capture of a scene.

Legal broker for the digital image property (optional)

This property encodes the name of the person or organization that holds the legal right to grant permissions or restrict use of the digital file.

Authorship property (optional)

This property encodes the name of the camera owner, photographer or image creator.

Intellectual property notes property (optional)

This property encodes additional information beyond the scope of other properties in this group.

6.4 Content Description Group

These properties describe the content of the image. Typically it is text that the user enters, either when the pictures are taken or later in the process. Table 6.5 lists the properties in this group.

Test target in the image property (optional)

This property encodes information about the type of scale or test target that is captured within the image frame. The values are in Table 6.6.

Group caption property (optional)

This property is text that describes the subject or purpose of a group of images (e.g., a roll of film). The image in the digital file is one member of the “group.”

Caption text property (optional)

This property is text that describes the subject or purpose of the image. It may be additionally used to provide any other type of information related to the image.

TABLE 6.5 **Properties in the content description group**

Property name	ID code	Type
Test target in the image	0x23000000	VT_UI4
Group caption	0x23000002	VT_LPWSTR
Caption text	0x23000003	VT_LPWSTR
People in the image	0x23000004	VT_LPWSTR | VT_VECTOR
Things in the image	0x23000007	VT_LPWSTR | VT_VECTOR
Date of the original image	0x2300000A	VT_FILETIME
Events in the image	0x2300000B	VT_LPWSTR | VT_VECTOR
Places in the image	0x2300000C	VT_LPWSTR | VT_VECTOR
Content description notes	0x2300000F	VT_LPWSTR

TABLE 6.6 **Valid test target in the image property values**

Value	Meaning
0x0	Unidentified
0x1	Color chart
0x2	Grey card
0x3	Greyscale
0x4	Resolution chart
0x5	Inch scale
0x6	Centimeter scale
0x7	Millimeter scale
0x8	Micrometer scale

People in the image property (optional)

This property encodes the personal or “role” names of people in the image. Personal names are any variation of FirstName, Initial, LastName, Titles of Address denotations (for example, Dr. Jane Smith). Roles may be occupational or situational denotations (for example, doctor). Multiple entries are allowed.

Things in the image property (optional)

This property encodes the names of tangible objects depicted in the image, (Washington Monument, for example). Multiple entries are allowed.

Date of the original image property (optional)

This property encodes the date and time the image was originally captured. In the case of a scanned photograph, this would be the date and time of the original photograph, not the date and time it was scanned. The date and time the digital file was created is stored in the property Scan Date. In the case of other printed materials, this would be the date the item was originally published.

Events in the image property (optional)
This property encodes the events depicted in the image. Events may be personal or societal (e.g., birthday, anniversary, New Year's Eve). Editorial applications may use this property to describe historical, political, or natural events (e.g., a coronation, the Crimean War, Hurricane Andrew).

Places in the image property (optional)
This property encodes the place depicted in the image (Chicago, Illinois). Multiple entries are allowed (e.g., the image may contain a map or an aerial view of a region).

Content description notes property (optional)
This property encodes additional user/application defined information beyond the scope of other properties in this group.

6.5 Camera Information Group

This group of properties describes the camera used to take a photograph. Table 6.7 lists the properties in this group.

TABLE 6.7 **Properties in the camera information group**

Property name	ID code	Type
Camera manufacturer name	0x24000000	VT_LPWSTR
Camera model name	0x24000001	VT_LPWSTR
Camera serial number	0x24000002	VT_LPWSTR

Camera manufacturer name property (optional)
This property encodes the name of the manufacturer or vendor of the camera or original-scene capture device.

Camera model name property (optional)
This property encodes the model name or number of the camera, and can include the serial number of the camera.

Camera serial number property (optional)
This property encodes the manufacturer's serial number of the camera as a text string.

6.6 Per Picture Camera Settings Group

This group of properties describes the camera settings used when the image was captured.

New generations of digital and film cameras make it possible to capture more information about the conditions under which a picture was taken. This may include information about the lens aperture and exposure time, whether a flash was used, which lens was used, etc. This technical information is useful to professional and serious amateur photographers. In addition, some of these properties are useful to image database applications for populating values useful to image analysis and retrieval. Table 6.8 lists the properties in this group.

TABLE 6.8 **Properties in the per picture camera settings group**

Property name	ID code	Type
Capture date	0x25000000	VT_FILETIME
Exposure time	0x25000001	VT_R4
F-number	0x25000002	VT_R4
Exposure program	0x25000003	VT_UI4
Brightness value	0x25000004	VT_R4 | VT_VECTOR
Exposure bias value	0x25000005	VT_R4
Subject distance	0x25000006	VT_R4 | VT_VECTOR
Metering mode	0x25000007	VT_UI4
Scene illuminant	0x25000008	VT_UI4
Focal length	0x25000009	VT_R4
Maximum aperture value	0x2500000A	VT_R4
Flash	0x2500000B	VT_UI4
Flash energy	0x2500000C	VT_R4
Flash return	0x2500000D	VT_UI4
Back light	0x2500000E	VT_UI4
Subject location	0x2500000F	VT_R4 | VT_VECTOR
Exposure index	0x25000010	VT_R4
Special effects optical filter	0x25000011	VT_UI4 | VT_VECTOR
Per picture notes	0x25000012	VT_LPWSTR

Capture date property (optional)

This property encodes the date and time the image was captured.

Exposure time property (optional)

This property encodes the exposure time used when the image was captured. The units are seconds.

F-number property (optional)

This property encodes the lens f-number (ratio of lens aperture to focal length) used when the image was captured.

Exposure program property (optional)

This property encodes the class of exposure program that the camera used at the time the image was captured. Typical exposure programs include normal-program (general-purpose auto-exposure), aperture-priority (user sets aperture, camera selects shutter speed to properly expose), shutter-priority (user sets shutter speed, camera selects aperture to properly expose), etc. Values greater than 0x8 must be handled by core reader software as though they were Unidentified (0x0). Possible values are listed in Table 6.9.

TABLE 6.9 **Valid exposure program property values**

Value	Meaning
0x0	Unidentified
0x1	Manual
0x2	Program normal
0x3	Aperture priority
0x4	Shutter priority
0x5	Program creative (biased toward greater depth of field)
0x6	Program action (biased toward faster shutter speed)
0x7	Portrait mode (intended for close-up photos with the background out of focus)
0x8	Landscape mode (intended for landscapes with the background in good focus)

Brightness value property (optional)

This property encodes the Brightness Value (BV) measured when the image was captured, using APEX units. The expected maximum value is approximately 13.00 corresponding to a picture taken of a snow scene on a sunny day, and the expected minimum value is approximately -3.00 corresponding to a night scene.

If the value supplied by the capture device represents a range of values rather than a single value, it is encoded as a VT_VECTOR of two VT_R4 real numbers. The first value represents the lower value of the range, and the second represents the higher value. If the capture device supplies an exact value, it is encoded as a VT_VECTOR with a single VT_R4 value in the vector.

Exposure bias value property (optional)

This property encodes the actual exposure bias (the amount of over- or under-exposure relative to a normal exposure, as determined by the camera's exposure system) used when capturing the image, using APEX units. The range is between -99.99 and 99.99.

The value is the number of exposure values (stops). For example, -1.00 indicates 1 eV (1 stop) underexposure, or half the normal exposure.

Subject distance property (optional)

This property encodes the distance (in meters) between the front nodal plane of the lens and the position at which the camera was focusing when the image was captured. Note that the camera may have focused on a subject within the scene which may not have been the primary subject.

If the value supplied by the capture device represents a range of values rather than a single value, it is encoded as a VT_VECTOR of two VT_R4 real numbers. The first value represents the lower value of the range, and the second represents the higher value. If the capture device supplies an exact value, it is encoded as a VT_VECTOR with a single VT_R4 value in the vector. Focus at infinity is encoded as -1.

Metering mode property (optional)

This property encodes the metering mode (the camera's method of spatially weighting the scene luminance values to determine the sensor exposure) used when capturing the image. Values greater than 0x4 must be handled by core reader software as though they were Unidentified (0x0). Possible values are listed in Table 6.10.

TABLE 6.10 **Valid metering mode property values**

Value	Meaning
0x0	Unidentified
0x1	Average
0x2	Center weighted average
0x3	Spot
0x4	Multi-spot

Scene illuminant property (optional)

This property encodes the light source (scene illuminant) that was present when the image was captured. Values between 0xB and 0x7FFF must be handled by core reader software as though they were Unidentified (0x0).

Note: Bit 15 of this 16-bit word is used as the key to whether or not a color temperature value is being stored. If bit 15 is 0, the value described within bits 0-14 will provide one of the prescribed color values depicted within the table below. If bit 15 is 1, bits 0-14 contain the actual color temperature value stored in units of Kelvin. In this case, color temperatures are limited to values in the range of 0 to 32767 Kelvin. Valid values are listed in Table 6.11. Values between 0xB and 0x7FFF must be handled by core reader software as though they were Unidentified (0x0).

Focal length property (optional)

This property encodes the lens focal length (in millimeters) used to capture the image.

Max aperture value property (optional)

This property encodes the maximum possible aperture opening (minimum lens f-number) of the camera or image capturing device, using APEX units. The allowed range is 1.00 to 99.99.

Flash property (optional)

This property encodes whether flash was used. Possible values are listed in Table 6.12. Values greater than 0x2 must be handled by core reader software as though they were Unidentified (0x0).

TABLE 6.11 **Valid scene illuminant property values**

Value	Meaning
0x0	Unidentified
0x1	Daylight
0x2	Fluorescent light
0x3	Tungsten lamp
0x4	Flash
0x5	Standard illuminant A
0x6	Standard illuminant B
0x7	Standard illuminant C
0x8	D55 illuminant
0x9	D65 illuminant
0xA	D75 illuminant
> 0x7FFF	The encoded actual temperature

TABLE 6.12 **Valid flash property values**

Value	Meaning
0x0	Unidentified
0x1	No flash used
0x2	Flash used

Flash energy property (optional)

This property encodes the amount of flash energy that was used. The measurement units are Beam Candle Power Seconds (BCPS).

Flash return property (optional)

This property encodes whether the camera judged that the flash was not effective at the time of exposure. Values greater than 0x2 must be handled by core reader software as though they were Unidentified (0x0). Possible values are listed in Table 6.13.

TABLE 6.13 **Valid flash return property values**

Value	Meaning
0x0	Unidentified
0x1	Subject outside flash range
0x2	Subject inside flash range

Back light property (optional)

This property encodes the camera's evaluation of the lighting conditions at the time of exposure. The definitions of the conditions are:

- Front lit: the subject is illuminated from the front side.
- Back lit 1: The brightness value difference between the subject center and the surrounding area is greater than one full step (APEX). The frame is exposed for the subject center.
- Back lit 2: The brightness value difference between the subject center and the surrounding area is greater than one full step (APEX). The frame is exposed for the surrounding area.

Values greater than 0x3 must be handled by core reader software as though they were Unidentified (0x0). Possible values are listed in Table 6.14.

TABLE 6.14 **Valid back light property values**

Value	Meaning
0x0	Unidentified
0x1	Front lit
0x2	Back lit 1
0x3	Back lit 2

Subject location property (optional)

This property identifies the approximate location of the subject in the scene. It provides an X column number and Y row number that corresponds to the center of the subject location. It is stored as a VT_VECTOR of two VT_R4 values, where the first value of the vector is the X location and the second value of the vector is the Y location, in resolution-independent coordinates where the height of the image is 1.0 and the width is the aspect ratio.

Exposure index property (optional)

This property encodes the exposure index setting the camera selected.

Special effects optical filter property (optional)

This property encodes the type of filter used. The property contains an array of filter values, where the order of the elements in the array indicates the stacking order of the filters. The first value in the array is the filter closest to the original scene. Possible values are listed in Table 6.15. Values greater than 0x7 must be handled by core reader software as though they were Unidentified (0x0).

Per picture camera settings notes property (optional)

This property encodes additional information not provided by the other properties. Both professional and amateur photographers may want to keep track of a variety of miscellaneous technical information, such as the use of extension tubes, bellows, close-up lenses, and other specialized accessories.

TABLE 6.15 **Valid special effects optical filter property values**

Value	Meaning
0x0	Unidentified
0x1	None
0x2	Colored
0x3	Diffusion
0x4	Multi-image
0x5	Polarizing
0x6	Split-field
0x7	Star

6.7 Digital Camera Characterization Group

This group of properties stores technical data specific to digital cameras. Table 6.16 lists the properties in the group.

TABLE 6.16 **Properties in the digital camera characterization group**

Property name	ID code	Type
Sensing method	0x26000000	VT_UI4
Focal plane X resolution	0x26000001	VT_R4
Focal plane Y resolution	0x26000002	VT_R4
Focal plane resolution unit	0x26000003	VT_UI4
Spatial frequency response	0x26000004	VT_VARIANT | VT_VECTOR
CFA pattern	0x26000005	VT_VARIANT | VT_VECTOR
Spectral sensitivity	0x26000006	VT_LPWSTR
ISO speed ratings	0x26000007	VT_UI2 | VT_VECTOR
OECF	0x26000008	VT_VARIANT | VT_VECTOR

Sensing method property (optional)

This property encodes the type of image sensor used in the camera or image capturing device. Possible values are listed in Table 6.17. Values greater than 0x8 must be handled by core reader software as though they were Unidentified (0x0).

TABLE 6.17 **Valid sensing method property values**

Value	Meaning
0x0	Undefined
0x1	Monochrome area sensor
0x2	One-chip color area sensor
0x3	Two-chip color area sensor
0x4	Three-chip color area sensor
0x5	Color sequential area sensor
0x6	Monochrome linear sensor
0x7	Trilinear sensor
0x8	Color sequential linear sensor

Focal plane X resolution property (optional)
This property encodes the number of pixels per *FocalPlaneResolutionUnit* in the ImageWidth direction for the main image. This property specifies the actual *FocalPlaneXResolution* at the focal plane of the camera. If this property is stored, the Focal length property in the per picture camera settings group must also be stored.

Focal plane Y resolution property (optional)
This property encodes the number of pixels per *FocalPlaneResolutionUnit* in the ImageLength direction for the main image. This property specifies the actual *FocalPlaneYResolution* at the focal plane of the camera. If this property is stored, the Focal length property in the per picture camera settings group must also be stored.

Focal plane resolution unit property (optional)
This property encodes the unit of measurement for the *FocalPlaneXResolution* and *FocalPlaneYResolution*. This property is mandatory if *FocalPlaneXResolution* or *FocalPlaneYResolution* exist. If this property is stored, the Focal length property in the per picture camera settings group must also be stored. Values other than those explicitly listed in Table 6.18 are not supported.

TABLE 6.18 **Valid focal plane resolution unit property values**

Value	Meaning
0x0	Inches
0x1	Meters
0x2	Centimeters
0x3	Millimeters

Spatial frequency response property (optional)

This property encodes the spatial frequency response (SFR) of the camera or image capturing device. The camera measured SFR data, described in ISO/TC42/WG18 Work Item [188] Working Draft 6.0, “Photography - Electronic still picture cameras - Resolution measurements,” can be stored as a table of spatial frequencies, horizontal SFR values, vertical SFR values, and diagonal SFR values. The following is a simple example of measured SFR data table (Table 6.19):

TABLE 6.19 **Sample frequency response**

Spatial frequency (lw/ph[a])	Horizontal SFR	Vertical SFR
0.1	1.00	1.00
0.2	0.90	0.95
0.3	0.80	0.85

a. line widths per picture height

The spatial frequency response is stored as a VT_VARIANT | VT_VECTOR in the format shown in Table 6.20.

TABLE 6.20 **Structure and entries of spatial frequency response VT_VARIANT | VT_VECTOR block**

Field	Type
Number of columns	VT_UI4
Number of rows	VT_UI4
Column headings	VT_LPWSTR | VT_VECTOR
Data	VT_R4 | VT_VECTOR

The number of entries in the column headings vector is the same as the number of columns, and the number of entries in the data field is the product of the number of rows and columns. Data entries are stored in row major order.

CFA pattern property (optional)

This property encodes the actual color filter array (CFA) geometric pattern of the image sensor used to capture a single-sensor color image. It is not relevant for all sensing methods.

The first value, *CFARepeatRows*, encodes the number of rows in the vertical direction needed to uniquely define the repeat pattern of the CFA. The second value, *CFARepeatCols*, encodes the number of columns in the horizontal direction that are needed to uniquely define the repeat pattern of the CFA. These two values are followed by a list of integer values of length (*CFARepeatRows* x *CFARepeatCols*) that define the color filter pattern, using the integers given in Table 6.21.

TABLE 6.21 **Valid CFA pattern property values**

Value	Meaning
0x0	Red
0x1	Green
0x2	Blue
0x3	Cyan
0x4	Magenta
0x5	Yellow
0x6	White

This property is stored in the form of a VT_VARIANT | VT_VECTOR, as shown in Table 6.20.

TABLE 6.22 **Structure and entries of CFA pattern VT_VARIANT | VT_VECTOR block**

Field	Type
CFARepeatRows	VT_UI2
CFARepeatCols	VT_UI2
CFAArray	VT_UI1 | VT_VECTOR

where *CFARepeatRows* and *CFARepeatCols* are the minimum number of rows and columns, respectively, needed to uniquely define the CFA pattern, and where *CFAArray* is a list of unsigned 1 byte integers, in row major order that define the pattern. For example, the property:

CFARepeatRow = 2
CFARepeatCol = 2
CFAArray = 1 0 2 1

corresponds to the Bayer CFA pattern shown below:

Line 0 = G R G R G R …
Line 1 = B G B G B G …
Line 2 = G R G R G R …
Line 1 = B G B G B G …

Spectral sensitivity property (optional)

This property field can be used to describe the spectral sensitivity of each channel of the camera used to capture the image. It is useful for certain scientific applications.

The property field is an ASCII string compatible with the “New Standard Practice for the Electronic Interchange of Color and Appearance Data” being developed within an ASTM Technical Committee. The ASCII string consists of a mandatory keyword list

followed by the associated data values. Mandatory keywords include NUMBER_OF_FIELDS, which equals the number of channels (spectral bands) + 1, and NUMBER_OF_SETS, which specifies the number of spectral frequency (wavelength) entries.

ISO speed ratings property (optional)

The property field is a VT_VECTOR of two VT_UI2 values. The first value is the ISO saturation speed rating classification and the second value is the ISO noise-based speed rating classification as defined in [21] tables 1 and 2.

OECF property (optional)

This property encodes the “Opto-Electronic Conversion Function” (OECF). The OECF is the relationship between the optical input and the image file code value outputs of an electronic camera. The property allows OECF values defined in [22] to be stored as a table of values. Table 6.23 shows a simple example of measured OECF data.

TABLE 6.23 **An example of measured OECF data**

Log exposure	Red output level	Green output level	Blue output level
-3.0	10.2	12.5	8.9
-2.0	48.1	47.5	48.3
-1.0	150.2	152.0	149.8

The OECF is stored as a VT_VARIANT | VT_VECTOR in the following format (Table 6.20).

TABLE 6.24 **Structure and entries of OECF VT_VARIANT | VT_VECTOR block**

Field	Type
Number of columns	VT_UI2
Number of rows	VT_UI2
Column headings	VT_LPWSTR | VT_VECTOR
Data	VT_R4 | VT_VECTOR

The number of entries in the column headings vector is the same as the number of columns, and the number of entries in the data field is the product of the number of rows and columns. Data entries are stored in row major order.

6.8 Film Description Group

This group of properties is used for images originating on photographic film. Table 6.25 lists the properties in the group.

TABLE 6.25 **Properties in the film description group**

Property name	ID code	Type
Film brand	0x27000000	VT_LPWSTR
Film category	0x27000001	VT_UI4
Film size	0x27000002	VT_VARIANT | VT_VECTOR
Film roll number	0x27000003	VT_UI4
Film frame number	0x27000004	VT_UI4

Film brand property (optional)

This property encodes the name of the film manufacturer, the brand name, product code and generation code (for example, Kodak Gold100, Kodak Aerial 100).

Film category property (optional)

This property encodes the category of film used. Legal values are listed in Table 6.26.

TABLE 6.26 **Valid film category property values**

Value	Meaning
0x0	Unidentified
0x1	Negative B/W
0x2	Negative color
0x3	Reversal B/W
0x4	Reversal color
0x5	Chromagenic
0x6	Internegative B/W
0x7	Internegative color

Values greater than 0x7 must be handled by core reader software as though they were Unidentified (0x0).

Note: Chromagenic refers to B/W negative film that is developed with a C41 process (i.e., color negative chemistry).

Film size property (optional)

This property encodes the size of the X and Y dimension of the film used, and the unit of measurement. These properties are encoded as VT_VARIANT | VT_VECTOR, and

internally consists of two VT_R4 dimensions and one VT_UI2 unit value indicator as shown in Table 6.27.

TABLE 6.27 **Structure and entries of original scanned image size VT_VARIANT | VT_VECTOR block**

Field	Type
Film size X	VT_R4
Film size Y	VT_R4
Film size unit	VT_UI2

Film size X and Y are the width and height of the original film used, respectively, represented in the unit specified by Film size unit. Film size unit has the same values as the focal plane resolution unit (Table 6.18).

Film roll number property (optional)
This property encodes the roll number of the film. For some film, this number is encoded on the film cartridge as a bar code.

Film frame number property (optional)
This property encodes the frame number from the roll of film.

6.9 Original Document Scan Description Group

This group of properties is used for images originating as documents or prints. Table 6.28 lists the properties in the group.

TABLE 6.28 **Properties in the original document scan description group**

Property name	ID code	Type
Original scanned image size	0x29000000	VT_VARIANT | VT_VECTOR
Original document size	0x29000001	VT_VARIANT | VT_VECTOR
Original medium	0x29000002	VT_UI4
Type of original	0x29000003	VT_UI4

Original scanned image size property (optional)
This property encodes the lengths of the X and Y dimension of the scanned area, and the unit of measurement. These properties are encoded as VT_VARIANT | VT_VECTOR, and internally consists of two VT_R4 dimensions and one VT_UI2 unit value indicator as shown in Table 6.29.

TABLE 6.29 **Structure and entries of original scanned image size VT_VARIANT | VT_VECTOR block**

Field	Type
Original size X	VT_R4
Original size Y	VT_R4
Original size unit	VT_UI2

Original size X and Y are the width and height of the original scanned image, respectively, represented in the unit specified by Original size unit. Original size unit has the same values as the focal plane resolution unit (Table 6.18).

Original document size property (optional)
This property encodes the lengths of the X and Y dimension of the original photograph or document, and the unit of measurement. These values are encoded as VT_VARIANT | VT_VECTOR, and internally consist of two VT_R4 dimensions and one VT_UI2 unit value indicator. It has the same format as the original scanned image size property (Table 6.29).

Original medium property (optional)
This property encodes the medium of the original photograph, document, or artifact. Possible values are shown in Table 6.30.

TABLE 6.30 **Valid original medium property values**

Value	Meaning
0x0	Unidentified
0x1	Continuous tone image
0x2	Halftone image
0x3	Line art

Type of reflection original property (optional)
This property encodes the type of the original document or photographic print. Possible values are shown in Table 6.31.

TABLE 6.31 **Valid type of reflection original property values**

Value	Meaning
0x0	Unidentified
0x1	B/W print
0x2	Color print
0x3	B/W document
0x4	Color document

6.10 Scan Device Property Group

This group of properties is used for images scanned from reflection prints, documents, photographic slides, or negatives. It contains the properties listed in Table 6.32.

TABLE 6.32 **Properties in the scan device property group**

Property name	ID code	Type
Scanner manufacturer name	0x28000000	VT_LPWSTR
Scanner model name	0x28000001	VT_LPWSTR
Scanner serial number	0x28000002	VT_LPWSTR
Scan software	0x28000003	VT_LPWSTR
Scan software revision date	0x28000004	VT_DATE
Service bureau/organization name	0x28000005	VT_LPWSTR
Scan operator ID	0x28000006	VT_LPWSTR
Scan date	0x28000008	VT_FILETIME
Last modified date	0x28000009	VT_FILETIME
Scanner pixel size	0x2800000A	VT_R4

Scanner manufacturer name property (optional)
This property encodes the manufacturer or vendor of the scanner.

Scanner model name property (optional)
This property encodes model name or number of the scanner. It can also include the serial number of the scanner.

Scanner serial number property (optional)
This property encodes the manufacturer's serial number of the scanner as a text string.

Scan software property (optional)
This property encodes the name and version of the scanner software or firmware.

Scan software revision date property (optional)
This property encodes the revision date of the scanner software or firmware. The date should be in GMT.

Service bureau/organization name property (optional)
This property encodes the name of the service bureau, photofinisher, or organization performing the scan.

Scan operator ID property (optional)
This property encodes a name or ID for the person operating the scanner.

Scan date property (optional)
This property encodes the date and time the image was originally captured and digitized. This property should never be changed after it is written in the image capture device.

Last modified date property (optional)
This property encodes the last modification date of the scanned data.

Scanner pixel size property (optional)
This property specifies the pixel size, in micrometers, of the scanner.

SECTION 7

FlashPix Image View Object

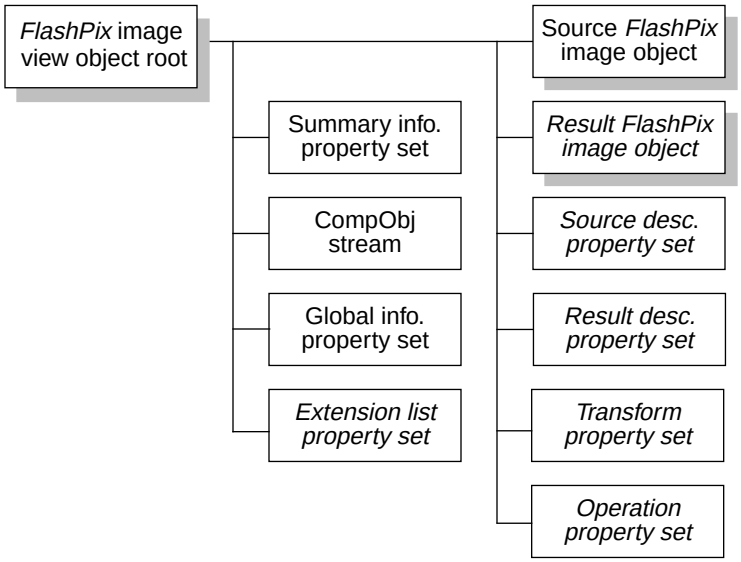
7.1 *FlashPix* Image View Object

The *FlashPix* format allows the specification of a viewing transform through a *FlashPix* image view object which references a *FlashPix* image. The viewing transform enables applications to represent a set of simple edits as a list of “commands” which are applied to the image in real time without altering the original image.

Storage name: Any valid storage name (recommend the file name in host file system)
Class ID: 56616700-C154-11CE-8553-00AA00A1F95B

This class ID is to be used for all *FlashPix* image view objects whether or not they contain any extensions. Figure 7.1 shows the storages and streams in a *FlashPix* image view. Those streams and storages in italics are optional or optional under certain circumstances.

FIGURE 7.1 ***FlashPix* image view storages and streams**



A *FlashPix* image view object contains the following storages and streams which are defined in more detail in subsequent sections. All property sets must abide by the restrictions specified in Section 1.4.1.

Source *FlashPix* image (required)
This storage is an instance of a *FlashPix* image object as defined in *Section 3: The FlashPix Image Object*. Transforms may be applied to the source (original) image to produce the result image (below).

Result *FlashPix* image (optional)
This storage is an instance of a *FlashPix* image object as defined in *Section 3: The FlashPix Image Object*. Transforms may be applied to the source image (above) to produce the result image.

Summary info property set (required)
This property set is an instance of the standard Summary Information property set, as described in Section 1.4.2. The viewing transform must be applied in creating the thumbnail image which is optional if the *FlashPix* file is authored with a non-hierarchical source image object in an embedded capture environment. It is otherwise required.

CompObj stream (required)
This stream contains the class ID of the *FlashPix* image view object.

Global info property set (required)
This property set contains generic information about the image view.

Source description property set (required)
This property set describes location and type of the source *FlashPix* image.

Result description property set (optional)
This property set describes the location and type of the result *FlashPix* images. It is required if the result *FlashPix* image object exists or if there is a viewing transform specified.

Transform property set (optional)
This property set describes the viewing transform specified for this image view.

Operation property set (optional)
This property set describes the actual view transform operation.

Extension list property set (optional)
This property set identifies extensions present in the *FlashPix* image view object including the data structures modified or added by each extension.

7.1.1 CompObj Stream (required)

The CompObj stream is a standard Structured Storage stream and is not a *FlashPix* stream. The header of the stream is not extended for storage of a stream class ID. This stream is required and is defined in Section A.3. The unicode versions of the CompObj stream fields are required.

The CompObj stream Clipboard Format field is used to store the class ID of the *FlashPix* image view object. The *FlashPix* image view object class ID is converted to a string for storage in the Clipboard Format field and must be bracketed by the bracket characters ‘{‘and ‘}’ just as returned by the OLE function StringFromGUID2.

The CompObj stream User Type field is generally used to store the User Type information from the OLE registry for the class ID. In OLE-enabled environments, the string contents should be retrieved from the OLE registry. In non-OLE-enabled environments, a string which is a user-understandable brief description of the object contents should be used.

The CompObj stream ProgID field is generally used to store the ProgID information from the OLE registry for the class ID. In OLE-enabled environments, the string contents should be retrieved from the OLE registry. In non-OLE-enabled environments, a string which identifies the program associated with the class ID should be used. This string cannot contain any spaces.

7.1.2 Source and Result *FlashPix* Image Objects

Names:	Data**040**Object**040**Store**040**%06d
Class ID:	56616000-C154-11CE-8553-00AA00A1F95B

The source and result *FlashPix* image objects are instances of the *FlashPix* image object as defined in *Section 3: The FlashPix Image Object*. The source *FlashPix* image object is required and the result *FlashPix* image object is optional. The single numeric parameter in the name represents the index of the source and result image objects. Upon creation, the index used must be unique in the *FlashPix* image view object. The maximum image index property of the global info property set must always be the maximum *FlashPix* image object index in the *FlashPix* image view.

The source image object represents the image to be processed through the viewing transform. For example, it may be an image that needs to be cropped and rotated or its color balance adjusted. The result image object is the image generated by applying the viewing transform to the source image object. The result image object cannot exist unless the *FlashPix* image view contains a viewing transform.

7.1.3 Source and Result Description Property Sets

Name:	**005**Data**040**Object**040**%06d
Class ID (for both):	56616080-C154-11CE-8553-00AA00A1F95B
Format ID:	56616080-C154-11CE-8553-00AA00A1F95B

These property sets have only one section, which has a format ID that is the same as the property set class ID.

These property sets are associated with the source and result image objects in this image view object. This association is indicated by matching index values in the *FlashPix* image object storage and description property set names.

Source description properties describe the source image object. Result description properties describe the result image object. Both property sets have the same format, as described below.

If the *FlashPix* image view does not contain a viewing transform, the result description property set is unused and may not exist.

The *FlashPix* image object to be used as the input to the image view must be characterized in the source description property set. After applying the viewing transform to the source image object, an actual *FlashPix* image may be stored in a result *FlashPix* image object. Even if the result image object is not stored, the result description property set must exist if there is a viewing transform to specify how the result is created.

The valid properties for these property sets are shown in Table 7.1.

Data object ID property (required)

This property specifies a unique ID used to identify the associated *FlashPix* image object. These values may be used, for example, in a networked system, to determine if a local copy of the images exist or if they must be pulled across the network.

TABLE 7.1 **Valid properties for the source and result description property sets**

Property name	ID code	Type
Data object ID	0x00010000	VT_CLSID
Locked property list	0x00010002	VT_UI4 | VT_VECTOR
Data object title	0x00010003	VT_LPWSTR
Last modifier	0x00010004	VT_LPWSTR
Revision number	0x00010005	VT_UI4
Creation time and date	0x00010006	VT_FILETIME
Modification time and date	0x00010007	VT_FILETIME
Creating application	0x00010008	VT_LPWSTR
Status	0x00010100	VT_UI4
Creator	0x00010101	VT_UI4
Users	0x00010102	VT_UI4 | VT_VECTOR
Cached image height	0x10000000	VT_UI4
Cached image width	0x10000001	VT_UI4

Locked property list property (optional)

This property specifies a list of properties that are locked. Each value in the list is taken as a property ID of a property found in this instance of the property set. If an editing application finds a property which is also found in the locked property list, it may not modify the value of the property. If the value of a locked property is modified, the results of rendering the *FlashPix* image view from another application will be undefined. If this property exists, it may not be deleted. This property is used to provide guidance to an editing application in situations where the *FlashPix* image view is a template to be “filled out” by the user.

Data object title (optional)

This property specifies a title for the associated image object. If this property exists, an editing application must keep the value updated.

Last modifier property (optional)

This property specifies the name of the last person (or system if the last modification was made by an automatic editing system) to modify the contents of the associated image object. If this property exists, an editing application must keep the value updated.

Revision number property (optional)

This property specifies the number of times the associated image object has been modified since its creation. If this property exists, an editing application must keep the value updated.

Creation time and date property (optional)

This property specifies the time and date of creation of the associated image object. If this property exists, an editing application must keep the value updated.

Modification time and date property (optional)
This property specifies the time and date of the last modification to the associated image object. If this property exists, an editing application must keep the value updated.

Creating application property (optional)
This property specifies the name of the application that created the associated image object. If this property exists, an editing application may not delete it.

Status property (required)
This property (Table 7.2) indicates the status of the value of the associated image. Possible values of existence/location field are shown in Table 7.3.

TABLE 7.2 Structure and entries of the status property

Field name	Length	Byte(s)
Existence data	2	0-1
Permissions set for data	2	2-3

TABLE 7.3 Valid status property values of the existence/location field

Value	Meaning
0x0	The data object associated with this property set does not exist.
0x1	The data object associated with this property set does exist.

The existence field indicates whether the associated image object exists or is stored for direct application access. If the existence field is 0x0 (not cached), the permissions field is ignored. The source image object must be cached (the value of the existence field must be 0x1). The result image object may or may not be cached, at the discretion of the writer. Possible values of the permissions field are shown in Table 7.4.

TABLE 7.4 Valid status property permissions field values

Value	Meaning
0x0	The data object is purgeable
0x1	The data object is not purgeable

If the associated image object is marked purgeable, a clean-up utility may delete the image object to recover storage space if it can be recreated from a source transform. Therefore, the source image object must be set to not purgeable (0x1). The result image object should be set to purgeable (0x0).

Creator property (required)

This property specifies the number of the transform node that created the image object. For the source image object, the creator should be set to zero (NULL). For the result image object, the creator should be set to the index of the viewing transform.

Users property (required)

This property specifies a list of transforms that take this image object as an input. Each entry in the list specifies the index of a transform. The array is unordered. If the associated image object is not used by any transforms, the number of elements in the array should be zero. Therefore, the users property must be zero for the result image object.

In the source description property set, the users property must be an array with one and only one element, the index used to name the transform property set. For the result description, a *FlashPix* image view writer should write an array with zero elements.

Cached image height and width properties (required if associated image exists)

These properties specify the height and width, respectively, of the image cached in the associated *FlashPix* image object. These properties are the height and width of the largest resolution in the associated *FlashPix* image object. These properties are required if the status property existence flag is set to cached (0x1). If the status property existence flag is set to not cached (0x0), these properties may not exist. Note that the height property precedes the width property contrary to other height and width property pairs in the format. Note also that the cached image height and width properties for the result *FlashPix* image object take precedence over the rectangle of interest and result aspect ratio properties should they exist in the transform property set.

7.1.4 Transform Property Set (optional)

Name:	**005**Transform**040**%06d
Class ID:	56616A00-C154-11CE-8553-00AA00A1F95B
Format ID:	56616A00-C154-11CE-8553-00AA00A1F95B

This property set describes the viewing transform specified for the image view. The single numeric parameter in the name represents the index of the transform. In a *FlashPix* image view, this array has only one element, the viewing transform. Upon creation, the index used must be unique in the *FlashPix* image view object. The maximum transform index property of the global info property set must always be the maximum transform index in use in the *FlashPix* image view object. The index is referenced by both the creator property of the result description property set and the users property of the source description property set, which must both have the same value. The transform property set is unused and must not exist if the *FlashPix* image view does not contain a viewing transform. Table 7.5 lists the possible properties of the transform list property set.

Transform node ID property (required)

This property specifies a unique ID used to identify the viewing transform. Note that this identifies the transform itself, not the value of the parameters. For example, this ID specifies that this transform node is performing the viewing transform on a particular

TABLE 7.5 **Valid properties for the transform property set**

Property name	ID code	Type
Transform node ID	0x00010000	VT_CLSID
Operation Class ID	0x00010001	VT_CLSID
Locked property list	0x00010002	VT_UI4 | VT_VECTOR
Transform title	0x00010003	VT_LPWSTR
Last modifier	0x00010004	VT_LPWSTR
Revision number	0x00010005	VT_UI4
Creation time and date	0x00010006	VT_FILETIME
Modification time and date	0x00010007	VT_FILETIME
Creating application	0x00010008	VT_LPWSTR
Input data object list	0x00010100	VT_UI4 | VT_VECTOR
Output data object list	0x00010101	VT_UI4 | VT_VECTOR
Operation number	0x00010102	VT_UI4
Result aspect ratio	0x10000000	VT_R4
Rectangle of interest	0x10000001	VT_R4 | VT_VECTOR
Filtering	0x10000002	VT_R4
Spatial orientation	0x10000003	VT_R4 | VT_VECTOR
Colortwist matrix	0x10000004	VT_R4 | VT_VECTOR
Contrast adjustment	0x10000005	VT_R4

source image. This ID does not change if the actual viewing parameters change (the transform is reexecuted).

Operation class ID property (required)

This property specifies the class ID of the operation to be performed by this transform node. This property (along with the class ID of this stream) specifies the code that actually executes the viewing transform. This property must have the value 56616A00-C154-11CE-8553-00AA00A1F95B.

Locked property list property (optional)

This property specifies a list of properties that are locked for the viewing transform. Each value in the list is a property ID of a property found in the transform property set. Editing applications may not modify the value of properties found in the locked property list. If the value of a locked property is modified, the results of rendering the *FlashPix* image view from another application will be undefined. If this property exists, it may not be deleted. This property is used to provide guidance to an editing application in situations where the *FlashPix* image view is a template to be “filled out” by the user.

Transform title property (optional)

This property specifies a title for the viewing transform. If this property exists, an editing application must keep the value updated.

Last modifier property (optional)
This property specifies the name of the last person (or system if the last modification was made by an automatic editing system) to modify the contents of the viewing transform. If this property exists, an editing application must keep the value updated.

Revision number property (optional)
This property specifies the number of times the viewing transform has been modified since its creation. If this property exists, an editing application must keep the value updated.

Creation time and date property (optional)
This property specifies the time and date of creation of the viewing transform. If this property exists, an editing application may not delete it.

Modification time and date property (optional)
This property specifies the time and date of the last modification to the viewing transform. If this property exists, an editing application must keep the value updated.

Creating application property (optional)
This property specifies the index of the application that created the viewing transform. If this property exists, an editing application must keep the value updated.

Input data object list property (required)
This property specifies the index used in naming the source *FlashPix* image that is input to the viewing transform. There may be only one element in the array.

Output data object list property (required)
This property specifies the index of the result image (in the sparse array of images). There may be only one element in the array.

Operation number property (required)
This property specifies the index used in naming the viewing operation.

Result aspect ratio property (optional)
The result aspect ratio property allows applications to specify the desired aspect ratio for image output. The value is an IEEE 4-byte floating point number.

The value (R) defines a rectangle with the top-left corner at (0,0) and the bottom-right corner at (R,1). The result aspect ratio must be applied to the output of the spatial orientation matrix as a cropping function. Pixels outside the rectangle are cropped and 100% transparent. If the property is not present, applications must operate as though the result aspect ratio is the same as the raw image aspect ratio. Note that the cached image height and width properties in the result *FlashPix* image object result description property set take precedence over the rectangle of interest and result aspect ratio properties.

Rectangle of interest property (optional)
The rectangle of interest property lets applications select part of the image. Only a rectangular region with sides parallel to the edges of the image can be specified. Each element of the rectangle of interest property array is an IEEE 4-byte floating point number. The format of the rectangle of interest property is given in Table 7.6. It is discussed in

more detail in Section 7.2.1. Applications must operate as though this property is defined with the array values (0,0,R,1) if the property is not present. This indicates that the entire image is selected. Note that the cached image height and width properties in

TABLE 7.6 **Format and fields of the rectangle of interest property**

Field	Length	Vector Element
left edge (*x*)	4	0
top edge (*y*)	4	1
width (*w*)	4	2
height (*h*)	4	3

the result *FlashPix* image object result description property set take precedence over the rectangle of interest and result aspect ratio properties.

Filtering property (optional)

The filtering property specifies the degree of filtering (sharpening / blurring) applied to the raw image data. The value is an IEEE 4-byte floating point number. The interpretation of the value is discussed in Section 7.2.2. Applications must operate as though this property has a zero value if it is not present to indicate that the raw image is not filtered.

Spatial orientation property (optional)

The spatial orientation property allows applications to rotate, flip, stretch, and shear an image. The value is an array of 16 IEEE 4-byte floating point numbers, with the first number starting at vector element 0. The position of the sixteen elements is as follows:

$$\begin{aligned} a_{ij} &\equiv \mathrm{ARRAY}[k] \\ k &= (i-1)\times 4 + j - 1 \end{aligned} \tag{7.1}$$

The interpretation of the value is discussed in Section 7.2.3. Applications must operate as though this property is defined with the array value (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1) if the property is not present. This indicates that there is no spatial transformation applied to the raw image.

Colortwist matrix property (optional)

The colortwist matrix property allows applications to make minor changes to the tone and color of a raw image. It is not intended to support correcting faults in the imaging chain. The value is an array of 16 IEEE 4-byte floating point numbers, with the first number starting at vector element 0. The position of the sixteen elements is as follows:

$$\begin{aligned} a_{ij} &\equiv \mathrm{ARRAY}[k] \\ k &= (i-1)\times 4 + j - 1 \end{aligned} \tag{7.2}$$

The interpretation of the value is discussed in Section 7.2.4. Applications must operate as though this property is defined with the array value (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1) if

the property is not present. This indicates that there is no tone or color correction applied to the raw image.

Contrast adjustment property (optional)

The contrast adjustment property allows applications to modify the contrast of a raw image. The value is an IEEE 4-byte floating point number. The interpretation of the value is discussed in Section 7.2.5. Applications must operate as though this property has a value of 1.0 if the property is not present. This indicates that there is no contrast adjustment applied to the raw image.

7.1.5 Operation Property Set (optional)

Name: **005**Operation**040**%06d
Class ID: 56616E00-C154-11CE-8553-00AA00A1F95B
Format ID: 56616E00-C154-11CE-8553-00AA00A1F95B

The operation property set specifies the software to execute the viewing transform. The single numeric parameter in the name represents the index of the operations in a sparse array. Upon creation, the index used must be greater than the maximum operation index property of the Global Info property set. In a *FlashPix* image view, the array has only one element, the viewing operation. The index is referenced by the operation number property in the transform property set.

If this *FlashPix* image view does not contain a viewing transform, the operation property set is unused and may not exist. Table 7.7 lists the possible properties for the operation property set.

TABLE 7.7 **Valid properties for the operation property set**

Property name	ID code	Type
Operation ID	0x00010000	VT_CLSID

Operation ID property (required)

This property specifies the class ID of the viewing operation. This value is used by either the *FlashPix* reader, an OLE server or an OpenDoc part to identify the actual software to implement the viewing transform. For a *FlashPix* image view, the value of this property must be 56616A00-C154-11CE-8553-00AA00A1F95B. This value specifies the actual code that executes the viewing transform.

7.1.6 Global Info Property Set (required)

Stream name: **005**Global**040**Info
Class ID: 56616F00-C154-11CE-8553-00AA00A1F95B
Format ID: 56616F00-C154-11CE-8553-00AA00A1F95B

This property set provides global information about the image view. Table 7.8 lists the possible properties for the global info property set.

TABLE 7.8 **Valid properties in the global info property set**

Property name	ID code	Type
Locked property list	0x00010002	VT_UI4 | VT_VECTOR
Transformed image title	0x00010003	VT_LPWSTR
Last modifier	0x00010004	VT_LPWSTR
Visible outputs	0x00010100	VT_UI4 | VT_VECTOR
Maximum image index	0x00010101	VT_UI4
Maximum transform index	0x00010102	VT_UI4
Maximum operation index	0x00010103	VT_UI4

Locked property list property (optional)

This property specifies a list of properties that are locked for this property set. Each value in the list is a property ID of a property found in this property set. Editing applications may not modify the value of properties found in the locked property list. If the value of a locked property is modified, the results of rendering the *FlashPix* image view from another application will be undefined. If this property exists, it may not be deleted. This property is used to provide guidance to an editing application in situations where the *FlashPix* image view is a template to be "filled out" by the user.

Transformed image title property (optional)

This property specifies a title for the viewing transform. If this property exists, an editing application must keep the value updated.

Last modifier property (optional)

This property specifies the name of the last person (or system if the last modification was made by an automatic editing system) to modify the contents of this *FlashPix* image view. If this property exists, an editing application must keep the value updated.

Visible outputs property (required)

This property specifies the output of the image view. The value of this property indicates the image that is to be considered the output of this file. There may be only one value in this array. If the *FlashPix* image view contains a viewing transform, this value must be the index used to name the result image. If a viewing transform is not specified, this value must be the index used to name the source image.

Maximum image, transform node, and operation index properties (required)

These properties specify the highest index in use for data objects, transforms, and operations. When an application creates a new entity, it is recommended that it use the value of the appropriate of these index properties + 1 as the index value for that entity. Then the appropriate index property must be updated if necessary so that it is the maximum of the indices in use for the type of entity. The values of these properties are 0 prior to creating any image, transform, and operation entities.

7.1.7 Extension List Property Set (optional)

Stream name: **005**Image**040**Contents
Class ID: 56616010-C154-11CE-8553-00AA00A1F95B
Format ID: 56616010-C154-11CE-8553-00AA00A1F95B

This property set identifies extensions present in the *FlashPix* image view object by class ID, name, and description as well as the data elements changed or added by each extension. The property set is optional, however, if the *FlashPix* image view object contains any extensions, the extension list property set must be present and all extensions, registered and private, in the *FlashPix* image view object must be described. The way in which the data associated with an extension is structured can take one or more of the following forms:

- New storage(s) may be added
- New stream(s) may be added
- New *FlashPix* stream(s) may be added
- New subimage(s) may be added to a *FlashPix* image object
- New property set(s) may be added
- New property(s) may be added to an existing property set section
- Element(s) may be added to core *FlashPix* property set vector properties that are defined as variable length
- Value of a core *FlashPix* stream field may be changed
- Value of a core property set property may be changed

There are five restrictions to structuring the data elements of an extension. First, new fields may not be added to existing *FlashPix* streams. Second, due to the inability to independently ensure property ID code uniqueness, only registered extensions may add properties to an existing property set section. Third, private extensions may not change the value of a core *FlashPix* stream field or a core property set property. Fourth, extensions can only add vector elements that are not already used by core or other extensions present in the file. Upon removal of an extension, the vector element values associated with the extension must be replaced with NULL and the vector must not be reordered. Fifth, only registered extensions can add elements to core property set vector properties.

Although there are a few practical examples where reasonable core reader actions could be defined for when an extension has changed the value of a core *FlashPix* stream field or a core property set property, these core reader actions must be considered in defining the core *FlashPix* specification. It is impractical to expect all core reader software to be updated to incorporate default actions identified in the course of developing new extensions. The definition of extensions must not impact the core definition unless some compelling feature set is identified which the *FlashPix* format Advisory Council agrees to include in a revised definition of the core *FlashPix* format. Therefore, efforts to define public extensions will avoid impacting core *FlashPix* stream field and core property set property values.

If the *FlashPix* image view object contains an extended *FlashPix* image object, the extended *FlashPix* image object must be listed in the extension list of the *FlashPix*

image view object. The details of how the *FlashPix* image object is extended are left to the extension list property set of the *FlashPix* image object itself. The extension list entry for the extended *FlashPix* image object must use the *FlashPix* image object class ID as the extension ID and the *FlashPix* image object storage name as the extension description.

If an extension is present in the *FlashPix* image view object which affects the output image appearance, an intermediate core *FlashPix* data object must be created as an intermediate source image object for core reader use. The creator transform of this source image object may not be a core viewing transform so it is clear to a core reader that this image object is truly its source and it will not attempt to resolve the creating transform. The intermediate source image object must be hierarchical, but is not required to be at the full resolution potential of images from which it is created. The extended authoring application may choose the resolution to make available. As core reader software cannot access data of a higher resolution than provided in the intermediate source image object, it is strongly recommended that data corresponding to at least 200dpi is provided. Further, the intermediate source image object does not have to be an exact representation of the output that is created from a reader supporting the extension. That may not be possible. Although the image content of the intermediate source image is also at the discretion of the authoring application, it is recommended that the closest feasible representation is provided.

The valid properties of the extension list property set are listed in Table 7.9. The

TABLE 7.9 **Valid properties for the extension property list property set**

Property name	ID code	Type
Used extension numbers	0x10000000	VT_UI2 | VT_VECTOR
Extension name	0x*iiii*0001	VT_LPWSTR
Extension class ID	0x*iiii*0002	VT_CLSID
Extension persistence	0x*iiii*0003	VT_UI2
Extension creation date	0x*iiii*0004	VT_FILETIME
Extension modification date	0x*iiii*0005	VT_FILETIME
Creating application	0x*iiii*0006	VT_LPWSTR
Extension description	0x*iiii*0007	VT_LPWSTR
Storage / stream pathname	0x*iiii*1000	VT_LPWSTR | VT_VECTOR
FlashPix stream pathname	0x*iiii*2000	VT_LPWSTR | VT_VECTOR
FlashPix stream field offset	0x*iiii*2001	VT_UI4 | VT_VECTOR
Property set pathname	0x*iiii*3000	VT_LPWSTR | VT_VECTOR
Property set ID codes	0x*iiii*3jj1	VT_LPWSTR | VT_VECTOR
Property vector elements	0x*iiii*3jj2	VT_LPWSTR | VT_VECTOR

extensions present in the *FlashPix* image view object are numbered for the convenience of grouping the descriptive information about each extension. Property ID codes

0xiiiixxxx describe the extension numbered 0x*iiii*.

Used extension numbers property (required)
This property lists all extension numbers *0xiiii* used in the extension list property set for the *FlashPix* image view object. The property value is an unordered array of *0xiiii* values.

All applications must update this property each time an extension is added to or removed from a *FlashPix* image view object.

Extension name property (required)
This property identifies the name of the extension. If the extension is registered, the name used must be that which is published in the official*FlashPix* Extension Specification. For private extensions, the name is the whatever short, descriptive label the authoring application chooses.

All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence.

Extension class ID property (required)
This property identifies a unique class ID for the extension. If the extension is registered, the class ID must be that which is published in the official*FlashPix* Extension Specification. For private extensions, the class ID is assigned by the authoring application.

All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence.

Extension persistence property (required)
This property identifies the persistence of the extension with respect to edits to the core data elements of the *FlashPix* image view object. The legal values for the extension persistence property are defined in Table 7.10.

It is the responsibility of the reader/writer application upon save or copy functions to retain the extension data elements by default, or in accordance with the extension persistence property.

All applications must retain this property upon a save or copy function by default, or in accordance with the extension persistence.

Extension creation date property (optional unless extension persistence property is 0x2)
This property specifies the time and date the authoring application added the extension to the *FlashPix* image view object. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

TABLE 7.10 **Legal values of the existence persistence property**

Value	Meaning
0x0	Extension is valid independent of core element edits
0x1	Extension is invalid upon core element edits
0x2	Extension is potentially invalid upon core element edits

Extension modification date property (optional unless extension persistence property is 0x2)

This property specifies the time and date of the last modification to the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Creating application property (optional)

This property specifies the name of the application that authored the extension in the file. If the property exists, any application editing the extension must update the value and all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Extension description property (optional)

The description property is a short (<80 character) description of the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence value.

Storage/stream pathname property (optional)

This property lists the full storage or non-*FlashPix* stream name, including the path in the structured storage file from the *FlashPix* image view object storage, for each storage or non-*FlashPix* stream the extension added to the *FlashPix* image view object. The path is specified using the standard Unix file specification tokens: "/" represents a directory separator and must be the first character of the property value. Wildcard characters “*” and “?” (where “*” matches any 0 or more characters and “?” matches any 1 character) are permitted in the path portion of the property value. If a storage is listed in the extension list property set, its contents should not also be listed as they are assumed to also be associated with that extension. If this property is omitted it is assumed that no storages are added to the *FlashPix* image view object for the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

FlashPix stream pathname property (optional)

This property lists the full *FlashPix* stream name, including the path from the *FlashPix* image view object storage, for each *FlashPix* stream the extension added to or modified in the *FlashPix* image view object. The path is specified using the standard Unix file specification tokens: "/" represents a directory separator and must be the first character of the property value. Wildcard characters “*” and “?” (where “*” matches any 0 or more characters and “?” matches any 1 character) are permitted in the path portion of

the property value. The array of values for the *FlashPix* stream pathname property and the *FlashPix* stream field offset property array of values for extension *iiii* are associated as described in Table 7.11. If this property is omitted it is assumed that no *FlashPix* streams are added to or modified in the *FlashPix* image view object for the extension. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

FlashPix stream field offset property (optional)

This property lists the byte offsets (after the header) into the *FlashPix* stream identified with the *FlashPix* stream pathname property array of fields modified by the extension. The array of values for the *FlashPix* stream field offset property and the *FlashPix* stream pathname property array of values for extension *0xiiii* are associated as described in Table 7.11. This property is required only if the *FlashPix* stream pathname property exists. If this property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

In the Table 7.11 example, there are two *FlashPix* stream data elements associated with

TABLE 7.11 **Example values of *FlashPix* stream identification**

Property	Index = 0	Index = 1
0x00172000	stream x pathname	stream y pathname
0x00172001	0xFFFFFFFF	64

extension 0x17. The first, at Index = 0, is an added *FlashPix* stream as there is a *FlashPix* stream pathname value but the *FlashPix* stream field offset is 0xFFFFFFFF. The second, at index 0xjj = 1, is a field in a core *FlashPix* stream who's value is not among those defined in the core *FlashPix* format. This is indicated by the presence of a non-0xFFFFFFFF *FlashPix* stream field offset value in addition to a *FlashPix* stream pathname value.

Property set pathname property (optional)

This property is an array that lists the full property set name, including the path from the *FlashPix* image object storage, for each property set the extension 0xiiii added, added to, or modified in the *FlashPix* image object. The path is specified using the standard Unix file specification tokens: "/" represents a directory separator and must be the first character of the property value. Wildcard characters "*" and "?" (where "*" matches any 0 or more characters and "?" matches any 1 character) are permitted in the path portion of the property value. Table 7.12 shows an example of how the property set pathname, property set ID codes, and property set vector elements for extension 0xiiii are associated. The array index of the property set pathname property corresponds to 0xjj in the properties 0xiiii3jj1 and 0xiiii3jj2.

This property set is optional and if omitted it is assumed that no property sets are added, added to, or modified in the *FlashPix* image object for the extensions. If the property exists all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

Property set ID codes property (optional)

This property lists the ID codes of properties which have been added to a core property set, or defined with non-core values by an extension to the *FlashPix* image object. The value of each array position of the property is a VT_LPWSTR that may be composed of comma separated values each of which are either an individual property ID code or hyphen-separated pair of property ID codes. The array of values for the property set ID codes and the property vector elements for particular property set 0xjj and extension 0xiiii are associated as described in Table 7.12. When a new property set is added by an extension, the property set ID codes property is not required. This property is required if an extension adds properties to a core property set or modifies core property set properties. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

Property vector elements property (optional)

Extensions can add vector elements to core properties that are defined as variable length vectors. This property lists the vector index for the values added to a particular vector property. The value of each array position of the property is a VT_LPWSTR that may be composed of comma separated values each of which are either an individual vector element or hyphen-separated pair of vector elements. The array of values for the property vector elements and the property set ID codes for a particular property set 0xjj and extension 0xiiii are associated as described in Table 7.12. This property is only required when an extension adds vector elements to a core property set property. If the vector elements property is present, and vector elements have not been added to its associated property set ID code(s), then the value of this property must be NULL. If the property exists, all applications must retain it upon save or copy functions by default, or in accordance with the extension persistence property value.

In the Table 7.12 example, there are three property sets associated with extension 0x19. The first index of the property 0x000193000, which corresponds to 0xjj=00, is a new property set being added by the extension as there is a property set pathname value, but the property set ID codes and property vector element properties for 0xjj=00 are not listed. The second index of the property 0x00193000, which corresponds to 0xjj=01, is a core property set in which property ID Codes 0x00011001-0x00011005 and 0x00001200 are being added by the extension. The third index of the property 0x00193000, which corresponds to 0xjj=02, is a core property set in which property ID code 0x00033000 is of type VT_VECTOR and the extension has added values in elements 3,4, and 5 of that vector. Property ID codes $00044001-$00044004 are new ID codes being added to the property set by the extension as well. In this case since the property ID codes are new, the value of 0x00193022 for this array position is assigned

to NULL. This example also shows that the extension has added a value in element 2 of both the vectors defined by existing property ID codes, 0x00055000 and 0x00066000.

TABLE 7.12 **Example values of property set identification**

Property	Index=0	Index=1	Index=2
000193000	PS x pathname (0xjj=00)	PS y pathname (0xjj=01)	PS z pathname (0xjj=02)
000193011	$000011001-$000011005, $000012000		
000193021	$000033000	$000044001-$000044004	$000055000, $000066000
000193022	3,4,5	NULL	2

7.2 Viewing Transform Parameters

The viewing transform parameters allow a view other than the raw image data itself. There are four classes of viewing parameters: selection, filtering, spatial orientation, and color reproduction. The application must provide a user interface that will help users work with and preview the viewing parameters.

7.2.1 Selection via Rectangle of Interest

The rectangle of interest property lets applications specify which part of the image to retain. Only a rectangular region with sides applications specify parallel to the edges of the image can be specified.

The rectangle of interest is specified in the rectangle of interest property in the transform property set as a horizontal, rectangular box. Four values are needed to specify the rectangle: left edge (x), top edge (y), width (w), and height (h). Each of these values is specified as a floating point number in the resolution-independent coordinate system described in Section 2.1.1.

The rectangle of interest is always interpreted in the context of a specific resolution layer. For example, if the layer has N pixels across (in x), numbered 0 to N-1, the range of columns is given by:

$$\lfloor N \times x + 0.5 \rfloor \leq \text{columns} \leq \lfloor N \times (x + w) + 0.5 \rfloor \qquad (7.3)$$

Note that while pixel locations should be used to identify a pixel neighborhood for spatial operations, the rectangle of interest box should run from the left edge of the unit square surrounding the first pixel to the right edge of the unit square surrounding the last pixel. This combination of parameters provides a rectangle of interest which is robust to

resolution changes. If the user selects a region manually at a specific resolution, the rectangle parameters should be calculated as follows, using the left edge as an example: calculate the location of the leftmost pixel in the continuous, resolution-dependent coordinate system described in Section 2.1.2. Scale this value to the resolution-independent system described inSection 2.1.1. This process will provide a stable description of the region when it is expressed at a higher or lower resolution.

The rectangle of interest does not imply any scaling or shifting into a "standard coordinate space." The rectangle only specifies that pixels outside the rectangle should be considered fully transparent. If the user desires that the area inside the rectangle of interest be displayed as "the whole image," the spatial orientation matrix (described in Section 7.2.3) should map the top-left corner of the rectangle to (0,0) and the bottom-right of the rectangle to (*w/h*,1).

7.2.2 Filtering

The *FlashPix* format viewing parameter that sharpens or blurs the image is referred to as image filtering. The degree of filtering is controlled by the filtering property in the transform property set. Positive values produce a sharper image; negative values produce a smoother, more blurry image, with less detail. A value of 0 leaves the image unchanged. Control is scaled so that one unit of filtering makes a just-noticeable change. Values between -20 and 20 may be expected to produce reasonable results. The default value of the filtering parameter is zero. It is stored as an IEEE 4-byte floating point number.

The influence of the filtering control is independent of image resolution. Proper operation of image filtering is closely tied to the resolution-independent rendering algorithms of the *FlashPix* format. When a reader requests actual *FlashPix* image data, it selects the best resolution layer to build the data from, and creates a digital filter to apply the requested degree of sharpening or blurring to that data.

To provide a degree of filtering independent of resolution, a reader also needs information about the imaging capabilities of the physical subsystems that generated the digital image and the subsystem that will be used to print the image. If this information is not available, a reader will use defaults that will yield good results in non-critical situations. Peripheral manufacturers could add value to systems that use the *FlashPix* format by providing this information with their devices (as part of their device driver or profiles).

7.2.2.1 The Measure

Filtering is defined as the change in the acutance of the complete imaging system. Acutance is a "one number" description of system sharpness. A number of definitions have been used, but in general acutance is a measure of the degree of blur introduced by all the elements of an imaging system, including the human eye that views the image. The only element missing from acutance is the degree of detail in the scene itself. Most modern measures of acutance are based on some integral of the overall MTF (modulation transfer function) of the system, including some eye MTF, scaled so that one unit of acutance change is a just-noticeable difference. Higher acutances are associated with

sharper imaging systems. A specific definition of acutance has been adopted for the *FlashPix* format.

The digital image is an intermediate result in the imaging chain that starts with image capture and ends with physical reproduction. Acutance is a measure of the ability of the chain to produce a sharp image. The customer can specify an increase or decrease in the acutance of this system. This will determine the digital filter used to increase or decrease the sharpness of the image.

7.2.2.2 Subsystem information

To precisely calculate the change in acutance of the system, some information about real devices is required. In particular, a reader requires an estimate of the capability of the input and output devices to reproduce fine image details. It also helps to have some estimate of how large the image will appear to the viewer, since the human eye is the final element in the imaging system.

Information on device sharpness is not generally available on the desktop today, even though manufacturers have measured it. An estimate of relative image size can often be made based on how the image is being used in a composition. In any case, the system is relatively insensitive to these factors, and the internal default values generally give good results.

Accurate information passed to a reader will improve the predictability and reliability of filtering as a change in acutance. Knowledge of these parameters also may permit a reader to work from a lower resolution level of the hierarchy, enabling faster rendering with no loss in image quality.

Four pieces of information may be utilized by a reader to determine acutance:

- A number describing the MTF (sharpness capability) of the capture operation sequence (stored in the sharpness approximation property of the file source group of the image info property set, Section 6.2)
- A number describing the MTF (sharpness capability) of the printer
- A number describing the MTF (blurring) of the prefilter used in building the *FlashPix* format hierarchy (stored for each resolution in the decimation prefilter width property of the resolution description groups of the image contents property set, Section 3.1.5.2).
- A number describing the relative size of the image as used by the customer

7.2.2.3 User Sharpening Adjustment

One challenge with providing image sharpening on the desktop is the difficulty novice users have in selecting the correct value. Generally, some sharpening is required to correct for the input/output devices, and some is added by the user for artistic effect. (A reader does not automatically add sharpening to correct for input and output devices, even when it has that information, but an application could do so, for example, by suggesting a filtering parameter setting.)

Handling sharpening adjustment for artistic effect is more difficult. However, the *FlashPix* filtering system can enable a very simple means for users to preview the sharpening effects if the subsystem information listed above is known. Specifically, numbers describing the MTFs of the printer and the monitor must be available.

7.2.3 Spatial Orientation

The *FlashPix* format provides spatial orientation parameters to rotate, scale, shear, and translate an image. These parameters are stated in terms of a 4×4 matrix, which maps image points from the displayed form of the image to the original image. In this form, the source points for resampling can be directly calculated.

Two-dimensional affine transformations are used to implement the spatial changes needed for image viewing. The general transform requires interpolation of the image data. A subset of the operations can be executed much faster because there is no change in the definition of the pixels (they only move around). It is the responsibility of the viewing engine to recognize these operations and to execute them efficiently.

The spatial orientation viewing parameters are specified by a 4×4 matrix as described by Equation 7.4:

$$\begin{bmatrix} x' \\ y' \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} \tag{7.4}$$

Each parameter is a floating point number. The 2-D affine transformations that map from the displayed form of the image to the original image are encoded in the six matrix elements a_{11}, a_{12}, a_{14}, a_{21}, a_{22}, a_{24}

In general, applying the affine transformations to the image will require resampling of the image. Note that the offset parameters a_{14} and a_{24} do matter for rotations and flips. In particular, rapid execution of these operations depends on certain values of the offsets.

The 4x4 matrix is stored, in the spatial orientation property of the Transform Property Set (Section 7.1.4), as an array of 16 IEEE 4-byte floating point numbers.

7.2.4 Tone and Color Corrections

The *FlashPix* format provides a colortwist matrix property in the transform property set to allow users to make small changes to the tone and color of an image. The changes are not designed to correct specific faults in the imaging chain, but to provide the user a simple way to specify common color corrections.

7.2.4.1 Color Images

Tone and color changes are applied in two ways. First, an affine matrix (colortwist matrix) on the vector (*Luma*, $Chroma_1$, $Chroma_2$, 1) of normalized PhotoYCC values is used to adjust lightness, saturation, and color balance. Many other kinds of simple changes can be effected with this matrix. The use of a matrix for this operation makes it easy to combine color operations done at different levels of an image composite. The matrix elements may be identified as follows:

$$\begin{bmatrix} Luma' \\ Chroma'_1 \\ Chroma'_2 \\ \end{bmatrix} = \begin{bmatrix} b_{YY} & b_{YC_1} & b_{YC_2} & b_{Yoff} \\ b_{C_1Y} & b_{C_1C_1} & b_{C_1C_2} & b_{C_1off} \\ b_{C_2Y} & b_{C_2C_1} & b_{C_2C_2} & b_{C_2off} \\ 0 & 0 & 0 & b_{off} \end{bmatrix} \begin{bmatrix} Luma \\ Chroma_1 \\ Chroma_2 \\ 1 \end{bmatrix} \tag{7.5}$$

Each element of the matrix is a floating point number. No limitation is placed on the matrix elements except for the left most three elements on the bottom row, which should all be zero. The input data to and the output data from the colortwist matrix MUST be normalized PhotoYCC. Although matrices for conversion into and out of normalized PhotoYCC may be concatenated with the colortwist matrix for efficiency, the colortwist matrix stored in the *FlashPix* image view object must NOT contain the matrix component for those conversions.The terms of the vector resulting from the matrix multiplication should be clipped to the correct limits for the desired output color space.

The values in the matrix are defined in terms of modifications to normalized PhotoYCC. Note that the tone and color correction matrix maps the source image to the destination image, which is different than the spatial orientation affine matrix.

7.2.4.2 Monochrome Images

Monochrome images should be treated as NIF RGB images where $R = G = B = X$. This allows a “tint” to be applied to a monochrome image using the viewing parameters, without requiring multichannel data to be stored in the *FlashPix* image.

7.2.5 Contrast adjustments

Contrast adjustments are controlled by the contrast adjustment property in the transform property set. It specifies a function through which the image data is passed. This operation may be implemented as a 1D lookup table or an equation applied to the image data once transformed to the proper color representation. The contrast change must be performed on an RGB representation of the image data which depends on the implementation method chosen. Additional (fixed) matrix operations (aside from the colortwist matrix) will be required to convert to and from the RGB values used in the contrast adjustment.

The first matrix implements the conversion from the input color space to normalized PhotoYCC where the colortwist matrix is applied. The matrices following the colortwist converts normalized PhotoYCC data to the proper RGB representation for application

of the contrast parameters. The choice of RGB representations includes normalized RGB(*rgb*) and a greater than 8 bit integer RGB_m representation. Once applied, some additional matrices are required to return the image data back to its pre-color/tone correction representation which can then be converted to a different output color space if desired.

The color and tone corrections can be applied to both NIFRGB and PhotoYCC data. While the colortwist matrix is applied to a normalized PhotoYCC data representation and contrast is applied in one of two RGB data representations, the matrices provided to do these conversions alone are not sufficient to convert from the NIFRGB color space to the PhotoYCC space or visa versa. They are sufficient to do the metric conversions required for the tone and color corrections, but are not for color space transformations. Therefore when the tone and color corrections are applied, the modified image data should be in the same color space as the original input color space at which point additional processing can be performed to convert the color space if needed. It may be possible, however, to concatenate some of the additional color space transformation processing with the color and tone correction matrices and LUTs to simplify the processing.

The desired contrast is specified by a single floating point number. A value of 1.0 indicates no contrast change; values greater than 1 provide higher contrast.

The contrast adjustment is stored as an IEEE 4-byte floating point number. Its default value is 1.0.

If the contrast parameter is other than 1.0 and the equation method for contrast is used, the procedure for implementing tone and color viewing parameters is as follows:

1. Combine a normalized PhotoYCC→normalized RGB conversion matrix with the given colortwist matrix and the input→normalized PhotoYCC matrix, producing a new matrix *M*'.
2. Pass the image data through *M*'
3. Apply the contrast modification through equation (7.6) to the image data resulting from *M*'.
4. Pass the contrast modified data through a series of matrices to convert the contrast modified normalized RGB back to the original color space which can then be converted to a different output space if needed.

If the contrast parameter is other than 1.0 and the LUT method for contrast is used, the procedure for implementing tone and color viewing parameters is as follows:

1. Combine a normalized PhotoYCC→normalized RGB conversion matrix and a normalized RGB→$RGB_m$with the given colortwist matrix and the input→normalized PhotoYCC matrix, producing a new matrix *M*'.
2. Pass the image data through *M*' limiting the RGB_m values to the range 0<=X<2M.
3. Create the LUT, K_{LUT}, using equation (7.7). The output of K_{LUT} is normalized RGB. If the RGB_m representation is preferred, cascade equation (7.8) with equation (7.7) to create K'_{LUT}.
4. Apply the contrast modification through either K_{LUT} or K'_{LUT}.

5. Pass the contrast modified data through a series of matrices to convert the contrast modified normalized RGB or RGB_m data back to the original color space which can then be converted to a different output space if needed.

For example, to increase the contrast by 20%, $K = 1.2$, where K is the contrast parameter. The following equations specify the contrast modification function, f_K, to be applied to *rgb* data:

$$p = 0.43$$

$$f_K(K, j) = \begin{cases} j < 0 & -p \times \left(\frac{-j}{p}\right)^K \\ j = 0 & 0 \\ 0 < j & p \times \left(\frac{j}{p}\right)^K \end{cases} \tag{7.6}$$

If the contrast modification is to be applied as a LUT, the input data must be converted to RGB_m space. The following equation can then be used to generate the LUT, K_{LUT}, where j' is the input pixel value in RGB_m space, where $M = 2^{k-1}$:

$$K_{LUT}[j'] = f_K\left(K, \frac{j' - \frac{M}{2}}{M}\right) \tag{7.7}$$

The output of K_{LUT} is *rgb* space. If RGB_m space is desired, the following equation can be cascaded with the equation for K_{LUT}, creating K'_{LUT}:

$$K'_{LUT}[j'] = K_{LUT}[j'] \times M + \frac{M}{2} \tag{7.8}$$

Note that if the “nine LUT method” is used for executing matrix multiplications, the 1D LUT and the final matrix can be combined into a single step.

7.3 Sequence of Viewing Parameters

The viewing parameters are not commutative. They must be applied in the following order: selection, filtering, spatial orientation, specification of result aspect ratio. The tone and color operation can be applied at any stage after the filtering step.

An application must carefully record edits to the image once it has been loaded. If the application wishes to modify the viewing parameters of the original image to reflect the edits, it must determine how to express the changes in terms of the allowed parameters.

7.3.1 Coordinate System

Both the selection and spatial orientation operations require clear specification of a coordinate system for the image. A *FlashPix* image can have any size, but the viewing parameters have no knowledge of the number of pixels in the image. The image is stored with an implicit orientation. The upper-left hand corner of the image has the coordinates (0, 0). The height of the image is 1.0, so that the lower-left corner has the coordinates (0, 1.0). The lower-right corner has the coordinates (*R*, 1.0), where *R* is the aspect ratio of the image.

The pixel locations are specified exactly for spatial operations: in a layer with *N* pixels in the x direction, the first pixel is centered at 0.5/*N*, the second is centered at 1.5/*N*, etc.

The affine spatial transform may rotate and shift the image data. Still, the coordinates implied by the specification of the result aspect ratio refer to the original coordinate system of the image.

For each pixel in the result rectangle (specified by the result aspect ratio), apply the affine transform to identify the corresponding location in the original image.

IF this point falls within the rectangle of interest,

THEN: This point will probably not fall in the middle of an original pixel. Use an interpolation scheme to calculate the values for the new pixel. Map these values through the tone and color transform to get the final image values.

ELSE: The point is outside of the defined image area. In the context of a viewer, nothing should be displayed—the viewer must define its background level. In the context of compositing, this pixel has 100% transparency.

7.3.2 Image Size and Limits

The *FlashPix* format is a resolution-independent format. There is no implicit scale to the image. The image should be displayed fully at whatever size and resolution suits the viewing device.

After a spatial orientation change leaves the image tilted or sheared, the resulting image is defined to be the area in the rectangle (0,0) to (R_{result},1), even though there may be legitimate image data outside this region. If the application wants the image to be rotated and displayed in its entirety, the affine matrix must also perform a scale to shrink the image such that it fits entirely inside the output rectangle.

APPENDIX A

Structured Storage

Note: This document is meant to accompany the Microsoft OLE Structured Storage Reference Implementation, hereafter referred to as the 'Software'. If this document and functionality of the Software conflict, the actual functionality of the Software represents the correct functionality. Microst assumes no responsibility for any damages that might occur either directly or indirectly from these discrepancies or inaccuracies. Microsoft may have trademarks, copyrights, patents or pending patent applications, or other intellectual property rights covering subject matter in this document and in the Software. The furnishing of this document does not give you a license to these trademarks, copyrights, patents, or other intellectual property rights and any license rights granted are limited to those set forth in the End User License Agreement accompanying this document.

A.1 Compound File Binary Format

A.1.0 Overview

A Compound File is made up of a number of **virtual streams**. These are collections of data that behave as a linear stream, although their on-disk format may be fragmented. Virtual streams can be user data, or they can be control structures used to maintain the file. Note that the file itself can also be considered a virtual stream.

All allocations of space within a Compound File are done in units called **sectors**. The size of a sector is definable at creation time of a Compound File, but for the purposes of this document will be 512 bytes. A virtual stream is made up of a sequence of sectors.

The Compound File uses several different types of sector: *Fat*, *Directory*, *Minifat*, *DIF*, and *Storage*. A separate type of 'sector' is a *Header*, the primary difference being that a Header is always 512 bytes long (regardless of the sector size of the rest of the file) and is always located at offset zero (0). With the exception of the header, sectors of any type can be placed anywhere within the file. The function of the various sector types is discussed below.

In the discussion below, the term **SECT** is used to describe the location of a sector within a virtual stream (in most cases this virtual stream is the file itself). Internally, a SECT is represented as a ULONG.

A.1.1 Sector Types

```
[4 bytes]   typedef unsigned long ULONG;
[2 bytes]   typedef unsigned short USHORT;
[2 bytes]   typedef short OFFSET;
[4 bytes]   typedef ULONG SECT;
[4 bytes]   typedef ULONG FSINDEX;
[2 bytes]   typedef USHORT FSOFFSET;
[4 bytes]   typedef ULONG DFSIGNATURE;
[1 byte]    typedef unsigned char BYTE;
[2 bytes]   typedef unsigned short WORD;
[4 bytes]   typedef unsigned long DWORD;
[2 bytes]   typedef WORD DFPROPTYPE;
[4 bytes]   typedef ULONG SID;
[16 bytes]  typedef CLSID GUID;

[8 bytes]   typedef struct tagFILETIME {
                                      DWORD dwLowDateTime;
                                      DWORD dwHighDateTime;
                                       } FILETIME, TIME_T;


[4 bytes]   const SECT DIFSECT= 0xFFFFFFFC;
[4 bytes]   const SECT FATSECT= 0xFFFFFFFD;
[4 bytes]   const SECT ENDOFCHAIN= 0xFFFFFFFE;
[4 bytes]   const SECT FREESECT= 0xFFFFFFFF;
```


A.1.1.1 Header

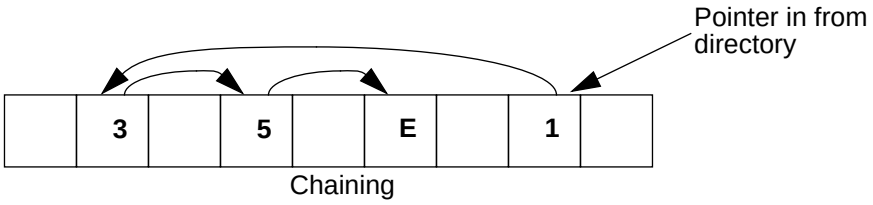
```
struct StructuredStorageHeader{// [offset from start in bytes, length
                               // in bytes]
    BYTE       _abSig[8];      // [000H,08] {0xd0, 0xcf, 0x11, 0xe0,
                               // 0xa1, 0xb1, 0x1a, 0xe1} for current
                               // version, was {0x0e, 0x11, 0xfc,
                               // 0x0d, 0xd0, 0xcf, 0x11, 0xe0} on old,
                               // beta 2 files (late '92) which are also
                               // supported by the reference
                               // implementation
    CLSID      _clid;          // [008H,16] class id (set with
                               // WriteClassStg, retrieved with
                               // GetClassFile/ReadClassStg)
    USHORT     _uMinorVersion; // [018H,02] minor version of the
                               // format: 33 is written by reference
                               // implementation
    USHORT     _uDllVersion;   // [01AH,02] major version of the dll/
                               // format: 3 is written by reference
                               // implementation
    USHORT     _uByteOrder;    // [01CH,02] 0xFFFE: indicates Intel
                               // byte-ordering
    USHORT     _uSectorShift;  // [01EH,02] size of sectors in power-
                               // of-two (typically 9, indicating 512-
                               // byte sectors)
    USHORT     _uMiniSectorShift; // [020H,02] size of mini-sectors
                               // in power-of-two (typically 6,
                               // indicating 64-byte mini-sectors)
    USHORT     _usReserved;    // [022H,02] reserved, must be zero
    ULONG      _ulReserved1;   // [024H,04] reserved, must be zero
    ULONG      _ulReserved2;   // [028H,04] reserved, must be zero
    FSINDEX    _csectFat;      // [02CH,04] number of SECTs in the FAT
                               // chain
    SECT       _sectDirStart;  // [030H,04] first SECT in the FAT
                               // Directory chain
    DFSIGNATURE_signature;     // [034H,04] signature used for transac
                               // tioning must be zero. The reference
                               // implementation does not support
                               // transactioning
    ULONG      _ulMiniSectorCutoff;// [038H,04] maximum size for
                               // mini-streams: typically 4096 bytes
    SECT       _sectMiniFatStart; // [03CH,04] first SECT in the
                               // mini-FAT chain
    FSINDEX    _csectMiniFat;  // [040H,04] number of SECTs in the
                               // mini-FAT chain
    SECT       _sectDifStart;  // [044H,04] first SECT in the DIF
                               // chain
    FSINDEX    _csectDif;      // [048H,04] number of SECTs in the DIF
                               // chain
    SECT       _sectFat[109];  // [04CH,436] the SECTs of the first
                               // 109 FAT sectors
    };
```

The *Header* contains vital information for the instantiation of a Compound File. Its total length is 512 bytes. There is exactly one *Header* in any Compound File, and it is always located beginning at offset zero in the file.

A.1.1.2 Fat Sectors

The **Fat** is the main allocator for space within a Compound File. Every sector in the file is represented within the Fat in some fashion, including those sectors that are unallocated (free). The Fat is a virtual stream made up of one or more Fat Sectors.

Fat sectors are arrays of SECTs that represent the allocation of space within the file. Each stream is represented in the Fat by a **chain**, in much the same fashion as a DOS file-allocation-table (FAT). To elaborate, the set of Fat Sectors can be considered together to be a single array -- each cell in that array contains the SECT of the next sector in the chain, and this SECT can be used as an index into the Fat array to continue along the chain. Special values are reserved for chain terminators (ENDOFCHAIN = 0xFFFFFFFE), free sectors (FREESECT = 0xFFFFFFFF), and sectors that contain storage for Fat Sectors (FATSECT = 0xFFFFFFFD) or DIF Sectors (DIFSECT = 0xFFFFFFC), which are not chained in the same way as the others.



The locations of Fat Sectors are read from the DIF (Double-indirect Fat), which is described below. The Fat is represented in itself, but not by a chain – a special reserved SECT value (FATSECT = 0xFFFFFFFD) is used to mark sectors allocated to the Fat.

A SECT can be converted into a byte offset into the file by using the following formula: SECT << ssheader._uSectorShift + sizeof(ssheader). This implies that sector 0 of the file begins at byte offset 512, not at 0.

A.1.1.3 MiniFat Sectors

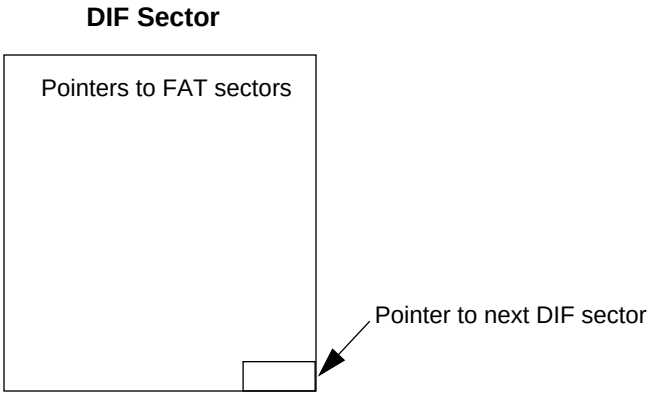
Since space for streams is always allocated in sector-sized blocks, there can be considerable waste when storing objects much smaller than sectors (typically 512 bytes). As a solution to this problem, we introduced the concept of the **MiniFat**. The MiniFat is structurally equivalent to the Fat, but is used in a different way. The virtual sector size for objects represented in the Minifat is 1 << ssheader._uMiniSectorShift (typically 64 bytes) instead of 1 << ssheader._uSectorShift (typically 512 bytes). The storage for these objects comes from a virtual stream within the Multistream (called the **Ministream**).

The locations for MiniFat sectors are stored in a standard chain in the Fat, with the beginning of the chain stored in the header.

A Minifat sector number can be converted into a byte offset into the ministream by using the following formula: SECT << ssheader._uMiniSectorShift. (This formula is different from the formula used to convert a SECT into a byte offset in the file, since no header is stored in the Ministream)

The Ministream is chained within the Fat in exactly the same fashion as any normal stream. It is referenced by the first Directory Entry (SID 0).

A.1.1.4 DIF Sectors



The **Double-Indirect Fat** is used to represent storage of the Fat. The DIF is also represented by an array of SECTs, and is chained by the terminating cell in each sector array (see the diagram above). As an optimization, the first 109 Fat Sectors are represented within the header itself, so no DIF sectors will be found in a small (< 7 MB) Compound File.

The DIF represents the Fat in a different manner than the Fat represents a chain. A given index into the DIF will contain the SECT of the Fat Sector found at that offset in the Fat virtual stream. For instance, index 3 in the DIF would contain the SECT for Sector #3 of the Fat.

The storage for DIF Sectors is reserved in the Fat, but is not chained there (space for it is reserved by a special SECT value , DIFSECT=0xFFFFFFFC). The location of the first DIF sector is stored in the header.

A value of ENDOFCHAIN=0xFFFFFFFE is stored in the pointer to the next DIF sector of the last DIF sector.

A.1.1.5 Directory Sectors

```
typedef enum tagSTGTY {
    STGTY_INVALID= 0,
    STGTY_STORAGE= 1,
    STGTY_STREAM= 2,
    STGTY_LOCKBYTES= 3,
    STGTY_PROPERTY= 4,
    STGTY_ROOT= 5,
    } STGTY;

typedef enum tagDECOLOR {
    DE_RED= 0,
    DE_BLACK= 1,
    } DECOLOR;

struct StructuredStorageDirectoryEntry {// [offset from start in bytes,
                                        // length in bytes]
    BYTE       _ab[32*sizeof(WCHAR)];   // [000H,64] 64 bytes. The
                                        // Element name in Unicode,
                                        // padded with zeros to fill
                                        // this byte array
    WORD       _cb;                     // [040H,02] Length of the
                                        // Element name in characters,
                                        // not bytes
    BYTE       _mse;                    // [042H,01] Type of object:
```



```
                                    // value taken from the STGTY
                                    // enumeration
    BYTE        _bflags;            // [043H,01] Value taken from
                                    // DECOLOR enumeration.
    SID         _sidLeftSib;        // [044H,04] SID of the left-
                                    // sibling of this entry in the
                                    // directory tree
    SID         _sidRightSib;       // [048H,04] SID of the right-
                                    // sibling of this entry in the
                                    // directory tree
    SID         _sidChild;          // [04CH,04] SID of the first
                                    // child acting as the root of
                                    // all the children of this el-
                                    // ement(if_mse=STGTY_STORAGE)
    GUID        _clsId;             // [050H,16]CLSID of this stor-
                                    // age (if_mse=STGTY_STORAGE)
    DWORD       _dwUserFlags;       // [060H,04] User flags of this
                                    // storage
                                    // (if_mse=STGTY_STORAGE)
    TIME        _T_time[2];         // [064H,16] Create/Modify
                                    // time-stamps
                                    // (if_mse=STGTY_STORAGE)
    SECT        _sectStart;         // [074H,04] starting SECT of
                                    // the stream
                                    // (if_mse=STGTY_STREAM)
    ULONG       _ulSize;            // [078H,04] size of stream in
                                    // bytes (if_mse=STGTY_STREAM)
    DFPROPTYPE _dptPropType;        // [07CH,02] Reserved for fu
                                    // ture use. Must be zero.
    };
```

The **Directory** is a structure used to contain per-stream information about the streams in a Compound File, as well as to maintain a tree-styled containment structure. It is a virtual stream made up of one or more Directory Sectors. The Directory is represented as a standard chain of sectors within the Fat. The first sector of the Directory chain (the Root Directory Entry)

Each level of the containment hierarchy (i.e. each set of siblings) is represented as a red-black tree. The parent of this set of sibilings will have a pointer to the top of this tree. This red-black tree must maintain the following conditions in order for it to be valid:

1. The root node must always be black.

2. No two consecutive nodes may both be red.

3. The left child must always be less than the right child. This relationship is defined as:

    - A node with a shorter name is less than a node with a longer name (i.e. compare the length of the name)
    - For nodes with the same length names, compare the two names.

The simplest implementation of the above invariants would be to mark every node as black, in which case the tree is simply a binary tree.

A Directory Sector is an array of Directory Entries, a structure represented in the diagram below. Each user stream within a Compound File is represented by a single Directory Entry. The Directory is considered as a large array of Directory Entries. It is useful to note that the Directory Entry for a stream remains at the same index in the Directory array for the life of the stream – thus, this index (called an **SID**) can be used to readily identify a given stream.

The directory entry is then padded out with zeros to make a total size of 128 bytes.

Directory entries are grouped into blocks of four to form Directory Sectors.

A.1.1.5.1 Root Directory Entry

The first sector of the Directory chain (also referred to as the first element of the Directory array, or SID 0) is known as the **Root Directory Entry** and is reserved for two purposes: First, it provides a root parent for all objects stationed at the root of the multi-stream. Second, its function is overloaded to store the size and starting sector for the Mini-stream.

The Root Directory Entry behaves as both a stream and a storage. All of the fields in the Directory Entry are valid for the root. The Root Directory Entry's Name field typically contains the string "RootEntry" in Unicode, although some versions of structured storage (particularly the preliminary reference implementation and the Macintosh version) store only the first letter of this string, "R" in the name. This string is always ignored, since the Root Directory Entry is known by its position at SID 0 rather than by its name, and its name is not otherwise used. New implementations should write "RootEntry" properly in the Root Directory Entry for consistency and support manipulating files created with only the "R" name.

A.1.1.5.2 Other Directory Entries

Non-root directory entries are marked as either stream (STGTY_STREAM) or storage (STGTY_STORAGE) elements. Storage elements have a _clsid, _time[ ], and _sidChild values; stream elements may not. Stream elements have valid _sectStart and _ulSize members, whereas these fields are set to zero for storage elements (except as noted above for the Root Directory Entry).

To determine the physical file location of actual stream data from a stream directory entry, it is necessary to determine which FAT (normal or mini) the stream exists within. Streams whose _ulSize member is less than the _ulMiniSectorCutoff value for the file exist in the ministream, and so the _startSect is used as an index into the MiniFat (which starts at _sectMiniFatStart) to track the chain of mini-sectors through the mini-stream (which is, as noted earlier, the standard (non-mini) stream referred to by the Root Directory Entry's _sectStart value). Streams whose _ulSize member is greater than the _ulMiniSectorCutoff value for the file exist as standard streams – their _sectStart value is used as an index into the standard FAT which describes the chain of full sectors containing their data).

A.1.1.6 Storage Sectors

Storage sectors are simply collections of arbitrary bytes. They are the building blocks of user streams, and no restrictions are imposed on their contents. Storage sectors are represented as chains in the Fat, and each storage chain (stream) will have a single Directory Entry associated with it.

A.1.2 Examples

This section contains a hexadecimal dump of an example structured storage file to clarify the binary file format.

A.1.2.1 Sector 0: Header

```
    _abSig     = DOCF 11E0 A1B1 1AE1
    _clid      = 0000 0000 0000 0000 0000 0000 0000 0000
    _uMinorVersion= 003B
    _uDllVersion= 3
    _uByteOrder= FFFE (Intel byte order)
    _uSectorShift= 9 (512 bytes)
    _uMiniSectorShift= 6 (64 bytes)
    _usReserved= 0000
    _ulReserved1= 00000000
    _ulReserved2= 00000000
    _csectFat  = 00000001
    _sectDirStart= 00000001
    _signature = 00000000
    _ulMiniSectorCutoff= 00001000 (4096 bytes)
    _sectMiniFatStart= 00000002
    _csectMiniFat= 00000001
    _sectDifStart= FFFFFFFE (no DIF, file is < 7Mb)
    _csectDIF  = 00000000
    _sectFat[] = 00000000 FFFFFFFF...(continues with FFFFFFFF)

000000:  D0CF 11E0 A1B1 1AE1  0000 0000 0000 0000   ................
000010:  0000 0000 0000 0000  3B00 0300 FEFF 0900   ................
000020:  0600 0000 0000 0000  0000 0000 0100 0000   ................
000030:  0100 0000 0000 0000  0010 0000 0200 0000   ................
000040:  0100 0000 FEFF FFFF  0000 0000 0000 0000   ................
000050:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
...
0001F0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```

A.1.2.2 SECT 0: First (Only) FAT Sector

```
SECT 0: FFFFFFFD = FATSECT: marks this sector as a FAT sector.
              Referred to in header by _sectFat[0]
SECT 1: FFFFFFFE = ENDOFCHAIN: marks the end of the directory chain,
              referred to in header by _sectDirStart
SECT 2: FFFFFFFE = ENDOFCHAIN: marks the end of the mini-fat, re
              ferred to in header by _sectMiniFatStart
SECT 3: 00000004 = pointer to the next sector in the “Stream 1” data.
              This sector is the first sector of “Stream 1”, it is re
              ferred to by the Directory Entry
SECT 4: ENDOFCHAIN (0xFFFFFFFE): marks the end of the “Stream 1”
              stream data. Further Entries are empty (FREESECT =
              0xFFFFFFFF)

000200:  FDFF FFFF FEFF FFFF  FEFF FFFF 0400 0000   ................
000210:  FEFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
...
0003F0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```


A.1.2.3 SECT 1: First (Only) Directory Sector

```
SID 0: Root SID: Root Name = "R"
SID 1: Element 1 SID: Name = "Storage 1"
SID 2: Element 2 SID: Name = "Stream 1"
SID 3: Unused
```

A.1.2.3.1 SID 0: Root Directory Entry

```
    _ab       = ("R")(this should be “Root Entry”)
    _cb       = 00042(42 bytes,does not include double-null termi
    nator)
    _mse      = 05 (STGTY_ROOT)
    _bflags   = 00 (DE_RED)
    _sidLeftSib= FFFFFFFF (none)
    _sidRightSib= FFFFFFFF (none)
    _sidChild  = 00000001 (SID 1: “Storage 1”)
    _clsid    = 0067 6156 54C1 CE11 8553 00AA 00A1 F95B
    _dwUserFlags= 00000000 (n/a for STGTY_ROOT)
    _time[0]   = CreateTime   = 0000 0000 0000 0000 (none set)
    _time[1]   = ModifyTime   = 801E 9213 4BB4 BA01 (??)
    _sectStart = 00000003 (starting sector of MiniStream)
    _ulSize   = 00000240 (length of MiniStream in bytes)
    _dptPropType= 0000 (n/a)

000400:  0052 0000 0000 0000  0000 0000 0000 0000   .R..............
000410:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000420:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000430:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000440:  04200 0500 FFFF FFFF  FFFF FFFF 0100 0000   ................
000450:  0067 6156 54C1 CE11  8553 00AA 00A1 F95B   .gaVT....S.....[
000460:  0000 0000 0000 0000  0000 0000 801E 9213   ................
000470:  4BB4 BA01 0300 0000  4002 0000 0000 0000   K.......@.......
```

A.1.2.3.2 SID 1: “Storage 1”

```
    _ab       = ("Storage 1")
    _cb       = 0014 (20 bytes, including double-null terminator)
    _mse      = 01 (STGTY_STORAGE)
    _bflags   = 01 (DE_BLACK)
    _sidLeftSib= FFFFFFFF (none)
    _sidRightSib= FFFFFFFF (none)
    _sidChild  = 00000002 (SID 2: “Stream 1”)
    _clsid    = 0000 0000 0000 0000 0000 0000 0000 0000 (none set)
    _dwUserFlags= 00000000 (none set)
    _time[0]   = CreateTime   = 00000000 00000000 (none set)
    _time[1]   = ModifyTime   = 00000000 00000000 (none set)
    _sectStart = 00000000 (n/a)
    _ulSize   = 00000000 (n/a)
    _dptPropType= 0000 (n/a)

000480:  5300 7400 6F00 7200  6100 6700 6500 2000   S.t.o.r.a.g.e. .
000490:  3100 0000 0000 0000  0000 0000 0000 0000   1...............
0004A0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0004B0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0004C0:  1400 0101 FFFF FFFF  FFFF FFFF 0200 0000   ................
0004D0:  0061 6156 54C1 CE11  8553 00AA 00A1 F95B   .aaVT....S.....[
0004E0:  0000 0000 0088 F912  4BB4 BA01 801E 9213   ........K.......
0004F0:  4BB4 BA01 0000 0000  0000 0000 0000 0000   K...............
```

A.1.2.3.3SID 2: “Stream 1”

```
    _ab        = ("Stream 1")
    _cb        = 0012 (18 bytes, including double-null terminator)
    _mse       = 02 (STGTY_STREAM)
    _bflags    = 01 (DE_BLACK)
    _sidLeftSib= FFFFFFFF (none)
    _sidRightSib= FFFFFFFF (none)
    _sidChild  = FFFFFFFF (n/a for STGTY_STREAM)
    _clsid     = 0000 0000 0000 0000 0000 0000 0000 0000 (n/a)
    _dwUserFlags= 00000000 (n/a)
    _time[0]   = CreateTime   = 00000000 00000000 (n/a)
    _time[1]   = ModifyTime   = 00000000 00000000 (n/a)
    _startSect = 00000000 (SECT in mini-fat, since _ulSize is
    smaller than _ulMiniSectorCutoff)
    _ulSize    = 00000220 (< ssheader._ulMiniSectorCutoff, so
    _sectStart is in Mini)
    _dptPropType= 0000 (n/a)

000500:  5300 7400 7200 6500  6100 6D00 2000 3100   S.t.r.e.a.m. .1.
000510:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000520:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000530:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000540:  1200 0201 FFFF FFFF  FFFF FFFF FFFF FFFF   ................
000550:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000560:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000570:  0000 0000 0000 0000  2002 0000 0000 0000   ........ .......
000580:  0000 0000 0000 0000  0000 0000 0000 0000   ................
```

A.1.2.3.4 SID 3: Unused

```
000590:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005A0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005B0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005C0:  0000 0000 FFFF FFFF  FFFF FFFF FFFF FFFF   ................
0005D0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005E0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005F0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
```

A.1.2.4 SECT 3: MiniFat Sector

```
    SECT 0: 00000001: pointer to the second sector in the “Stream 1”
    data. This sector is the first sector of “Stream 1”,
    it is referred to by _sectStart of SID 2
    SECT 1: 00000002: pointer to the third sector in the “Stream 1”
    data. This sector is the second sector of “Stream 1”,
    it is referred to in MiniFat SECT 0, above.
...
    SECT 8: FFFFFFFE = ENDOFCHAIN: marks the end of the “Stream 1”
    data.

    Further Entries are empty (FREESECT = 0xFFFFFFFF)

000600:  0100 0000 0200 0000  0300 0000 0400 0000   ................
000610:  0500 0000 0600 0000  0700 0000 0800 0000   ................
000620:  FEFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
...
0007F0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```


A.1.2.5 SECT 4: MiniStream (Data of “Stream 1”)

```
// referred to by SECTs in MiniFat of SECT 3, above

000800:  4461 7461 2066 6F72  2073 7472 6561 6D20   Data for stream
000810:  3144 6174 6120 666F  7220 7374 7265 616D   1Data for stream
000820:  2031 4461 7461 2066  6F72 2073 7472 6561    1Data for strea
...
000A00:  7461 2066 6F72 2073  7472 6561 6D20 3144   ta for stream 1D
000A10:  6174 6120 666F 7220  7374 7265 616D 2031   ata for stream 1

// data ends at 000A1F, MiniSector is filled to the end with known data
// (a copy of the header or FFFFFFF to prevent random disk or memory
// contents from contaminating the file on-disk.

000A20:  0000 0000 0000 0000  3B00 03FF FE00 0900   ........;.......
000A30:  0600 0000 0000 0000  0000 0000 0000 0100   ................
000A40:  D0CF 11E0 A1B1 1AE1  0000 0000 0000 0000   ................
000A50:  0000 0000 0000 0000  003B 0003 FFFE 0009   .........;......
000A60:  0006 0000 0000 0000  0000 0000 0000 0001   ................
000A70:  0000 0001 0000 0000  0000 1000 0000 0002   ................
000A80:  0000 0001 FFFF FFFE  0000 0000 0000 0000   ................
000A90:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
...
000BF0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```

A.2 OLE Property Set Binary Format

A.2.0 Document Properties in Storage

In an IStorage, a serialized property set is stored in either a single stream or in a nested IStorage instance. In the latter case, the contained stream named "Contents" is the primary stream containing property values. The format of the primary stream, the same in either case, is described in the next section below. None of the property types VT_STREAM, VT_STORAGE, VT_STREAMED_OBJECT, or VT_STORED_OBJECT may be used in a stream-based property set; these types may only be used in storage-based sets. It is the person who invents / defines a new property set who gets to choose whether the set is always stream-based, is always storage-based, or at times can be either.

Names in an IStorage that begin with the value '\0x05' are reserved exclusively for the storage of property sets. Streams or storages that begin with '\0x05' must therefore be in the format described below; storages so named must contain a "Contents" stream in the format.[1] One of the things that a person who invents a new standard property set does is specify the standard string name under which instances of that type are stored. For example, the summary information property set defined by OLE2 is always found under the name "\005SummaryInformation". OLE2 provided no conventions for choosing this name; however, a convention for choosing such names is now strongly recommended below.

Primary stream of a serialized property set

Property Set Header

Byte Order Indicator (WORD)	Format Version (WORD)	Originating OS Version (DWORD)	Class Identifier (CLSID)	Reserved (DWORD)

FMTID/Offset Pair

FMTID (16 bytes)	Offset* (DWORD)

Section

Section Header

Size of Section (DWORD)	Count of Properties, m (DWORD)

Property ID/Offset Pairs

Property ID for Property 1 (DWORD) Property ID for Property 2 (DWORD)	Offset** (DWORD) Offset** (DWORD)
m entries	
Property ID for Property m (DWORD)	Offset** (DWORD)

Properties (Type/Value Pairs)

Type Indicator 1 (DWORD) Type Indicator 2 (DWORD)	Property Value 1 (Variable Length) Property Value 2 (Variable Length)
m entries	
Type Indicator m (DWORD)	Property Value m (Variable Length)

*Offset in bytes from the start of the stream to the start of the section
**Offset in bytes from the start of the section to the start of the type/value pair

Figure 1. Stream containing a serialized property set

1. Properties may of course be stored in streams or storages that do not begin with '\0x05', but such properties are completely private to the application manipulating the storage; there is little reason to do this.

A.2.1 Format of the primary property set stream

The overall structure of a stream containing a serialized property set is as illustrated in Figure 2. The format consists of a property set header, a sequence of size exactly one of format id / offset pair, and a corresponding sequence of sections containing the actual property values.[1]

Absolutely all the fields of a serialized property set specified here are *always* stored in storage in little-endian (Intel) byte order.[2]

The overall length of this property set stream is limited to 256k bytes.

A.2.1.1 Property Set Header

At the beginning of the property set stream is a header. The following structure illustrates the header:

```
typedef struct PROPERTYSETHEADER {
                                    WORD   wByteOrder;// Always 0xFFFE
                                    WORD   wFormat;// Should be 0
                                    DWORD  dwOSVer;// System version
                                    CLSID  clsid; // Application CLSID
                                    DWORD  reserved;// Should be 1
                                    } PROPERTYSETHEADER;
```

The definition of the members of this structure as as follows.

Member	Meaning
wByteOrder	The byte-order indicator is a WORD and should always hold the value 0xFFFE. This is the same as the Unicode© byte-order indicator. When written in little-endian (Intel) byte order, as is always done, this appears in the stream as 0xFE, 0xFF.
wFormat	The format version is a WORD and indicates the format version of this stream. Property set writers should write zero for this value. Property set readers should check this value; if it is non-zero, then they should refuse to read the set, for it is in a format that they don't in fact understand.
dwOSVer	The OS version number is encoded as OS kind in the high order word (0 for Windows on DOS, 1 for Macintosh, 2 for Windows 32-bit, 3 for UNIX) and the OS-supplied version number in the low order word. For Windows on DOS and Windows 32-bit, the latter is the low order word of the result of GetVersion().
clsid	The class identifier is the CLSID of a class that can display and/or provide programmatic access to the property values. If there is no such class, it is recommended that the Format ID be used (see below), though a value of all zeros is also acceptable; the former simply allows for greater future extensibility.
reserved	Reserved for future use. A writer of a property set should write the value one here; a reader of a property set should only however check that the value is at least one.

1. The original OLE2 format allowed for more than one section, but use of that functionality is discourarged and no longer supported.

2. Notwithstanding the fact that there is a byte-order tag of 0xFFFE at the start of the format. This tag was intended to allow for future extensibility that has been subsequently determined to be very unlikely to be done.

A.2.1.2 Format ID / Offset Pairs

This part of the serialized property set indicates two things: the FMTID that scopes the property values contained in the set, and the location within the stream at which those values are stored.

```
typedef struct FORMATIDOFFSET {
                                FMTID  fmtid; // semantic name of a section
                                DWORD  dwOffset;// offset from start of whole
                                         //property set stream to the
                                         //section
                                } FORMATIDOFFSET;
```

The offset is the distance of bytes from the start of the whole stream to where the section begins. The format id (FMTID) is the semantic name of its corresponding section, telling how to interpret the property values therein.

A.2.1.3 Sections

Each section is made of up a property section header followed by an array that locates each property value within the section. It is specifically *not* the case that the properties in this array are sorted in any particular order Offsets within this array are the distance from the start of the section to the start of the property (type, value) pair. This allows entire sections to be copied as an array of bytes without any translation of internal structure.

```
typedef struct PROPERTYSECTIONHEADER {
                                        DWORD  cbSection;// size of section in
                                                 //bytes, which is
                                                 //inclusive of the byte
                                                 //count itself
                                        DWORD  cProperties;// count of properties
                                                  //in section
                                        PROPERTYIDOFFSETrgprop[];// array of
                                                        //property
                                                        //locations
                                             } PROPERTYSECTIONHEADER;

typedef struct PROPERTYIDOFFSET {
                                        DWORD  propid;// name of a property
                                        DWORD  dwOffset;// offset from the start
                                                 //of the section to that
                                                 //property
                                               } PROPERTYIDOFFSET;
```

Each property value contains a type tag followed by the bytes of the actual property value (at last!). All type/value pairs begin on a 32-bit boundary. Thus values may be followed with null bytes to align the subsequent pair on a 32-bit boundary (note though that there is no guarantee that property values are in fact as tightly packed in a section as this restriction permits; that is, there may be additional gratuitous padding).

```
typedef struct SERIALIZEDPROPERTYVALUE {
                                        DWORD  dwType;// type tag
                                        BYTE   rgb[]; // the actual property
                                                //value
                                           } SERIALIZEDPROPERTYVALUE;
```

A consequence of these rules is that the smallest legal section, one containing zero properties, contains the following eight bytes: 08 00 00 00 00 00 00 00.

A.2.2 Special property ids

A couple of property ids have special significance in all property sets.

A.2.2.1 Property Id zero: Dictionary of property names

To enable users of property sets to attach meaning to properties beyond those provided by the type indicator, property id zero is reserved in all property sets for an optional dictionary giving human readable names for the properties in the set and for the property set itself. The value will be an array of (property id, string) pairs.

The value of property id zero is an array of propid / string pairs. Entries in the array are the ids and corresponding names of the properties; these are not in any particular order with respect to their property ids. Not all of the names of the properties in the set need appear in the dictionary: the dictionary may omit entries for properties that are assumed to be universally known by clients that manipulate the property set. Typically names for the base property sets for widely accepted standards will be omitted.

Property names that begin with the binary Unicode characters 0x0001 through 0x001F are reserved for future use.

The name indicated as corresponding to property id zero is to be interpreted as the human readable name of the property set itself; like all property names, this may or may not be present.

The dictionary is stored as a list of Property ID/string pairs; the code page for the strings involved is as indicated in property id one. This can be illustrated using the following pseudo-structure definition for a dictionary entry (it's a pseudo-structure because the sz[] member is variable size).

```
 typedef struct tagENTRY {
                            DWORD    propid;    // Property ID
                            DWORD    cb;        // Count of bytes in the string,
                                                //including the null at the end
                            tchar    tsz[cb];   // Zero-terminated string. Code
                                                //page as indicated by property id
                                                //one.
                                    } ENTRY;

typedef struct tagDICTIONARY {
                                 DWORD    cEntries;  // Count of entries in the list
                                 ENTRY    rgEntry[cEntries];
                                          } DICTIONARY;
```


Note the following:

- Property ID zero does not have a type indicator. The DWORD that indicates the count of entries sits in the usual type indicator position.
- The count of bytes in the string (cb) includes the zero character that terminates the string.
- If the code page indicator is not 1200 (Unicode), there is no padding between entries to achieve reasonable alignment (sigh). However, if the code page indicator is Unicode, then each entry should be aligned on a DWORD boundary.
- If the code page indicator is not 1200 (Unicode), property names are stored dbcs strings. If the code page indicator does indicate Unicode, property name strings are stored as Unicode.
- Property name strings are restricted in length to 128 characters including the NULL terminating character.

A.2.2.2 Property Id One: Code Page Indicator

Property id one (1) is reserved as an indicator of which code page or script any not-always-Unicode strings in the property set originated from (code pages are used in Windows and scripts are from the Macintosh world). All such string values in the entire property set, such as VT_LPSTRs, VT_BSTRs, and the names in the property name dictionary found in code page zero use characters from this one code page. If the code page indicator is not present, the prevailing code page on the reader's machine must be assumed. If an application cannot understand the indicated code page, it should not try to modify strings stored in the property set.

When an application that is not the author of a property set changes a property of type string in the set, it should examine the code page indicator and take one of the following courses of action:

1. Write the new value using the code page found in the code page indicator.

2. Rewrite all string values in the property set using the new code page (including the new value), and modify the code page indicator to reflect the new code page.

Possible values for the code page indicator are given in the Win32 API reference (see the NLSAPI functions, and specifically the GetACP function) and Inside Macintosh Volume VI, §14-111. For example, the code page US ANSI is represented by 0x04e4 (or 1252 in decimal); the code page for Unicode is 1200. Whether a Windows code page or a Macintosh script is found in property id one is determinted by the "originating OS version" (PROPERTYSETHEADER::dwOSVer) of the property set as a whole. Note that there exist Windows code page equivalents for the Macintosh scripts numbers (Windows code page 10000, for example, is the Macintosh Roman script).

By far, if it is at all possible, it is recommended that the Unicode code page (1200) be used. This is the only practical way to in fact achieve worldwide interoperable property sets. In code page 1200, note especially that the count at the start of a VT_LPSTR or VT_BSTR is to be interpreted as a *byte* count, not a character count. The byte count includes the two zero bytes at the end of the string.

Property id one is of type VT_I2, and therefore consists of a DWORD containing VT_I2 followed by a USHORT indicating the code page. For example, the type/value pair for property ID one representing the US ANSI code page is the following six bytes:

02 00 00 00 e4 04

plus any necessary padding.

A.2.2.3 Property Id 0x80000000: Locale Indicator

Property Id 0x80000000 (PID_LOCALE) is reserved as an indication of which locale the property set was written in. The default locale for a property set, in the event that PID_LOCALE does not exist in the property set will be the system's default locale (LOCALE_SYSTEM_DEFAULT).

Applications can choose to support locale or just get the default behaviour. Applications that allow users to specify a working locale should write that locale identifier to this property. Applications that use the user's default locale (LOCALE_USER_DEFAULT) should write the user's default locale identifier.

Applications should be concerned with the possibility of getting information from a property set which is of a different locale than the app's locale or the user's or the system's (i.e. a foreign object).

There is no provision in the OLE Property Set interfaces defined above to specifically read and write PID_LOCALE; in other words this property can be treated just like any property. Likewise the system will not attempt to automatically add or modify this property.

Property Id PID_LOCALE is of type VT_U4, and therefore consists of a DWORD containing VT_U4 followed by a DWORD containing the Locale Identifier (LCID) as defined by Appendix C of the Win32 SDK.

A.2.2.4 Reserved property ids

Property ids with the high bit set (that is, which are negative) are reserved for future definition by Microsoft.

A.2.3 Property Type Representations

A property (type, value) pair is a DWORD type indicator, followed by a value whose representation depends on the type. The serialized representations of each of the different types of values are as follows:

Type indicator	**Value Representation**
VT_EMPTY	no bytes
VT_NULL	no bytes
VT_I2	2 byte signed integer
VT_I4	4 byte signed integer
VT_R4	32bit IEEE Floating point value
VT_R8	64bit IEEE Floating point value
VT_CY	8 byte two's complement integer (scaled by 10,000)
VT_DATE	A 64bit floating point number representing the number of days (not seconds) since December 31, 1899 (thus, January 1, 1900 is 2.0, January 2, 1900 is 3.0, and so on). This is stored in the same representation as VT_R8.

VT_BSTR	Counted, null terminated binary string; represented as a DWORD byte count of the number of bytes in the string (including the terminating null) followed by the bytes of the string. Character set is as indicated by the code page indicator.
VT_ERROR	A DWORD containing a status code.
VT_BOOL	Boolean value, a WORD containing 0 (false) or -1 (true).
VT_VARIANT	A type indicator (a DWORD) followed by the corresponding value. VT_VARIANT is only used in conjunction with VT_VECTOR: see below.
VT_UI1	1 byte unsigned integer
VT_UI2	2 byte unsigned integer
VT_UI4	4 byte unsigned integer
VT_I8	8 byte signed integer
VT_UI8	8 byte unsigned integer
VT_LPSTR	This is the representation of many strings. Stored in the same representation as VT_BSTR. Note therefore that the serialized representation of VT_LPSTR in fact has a preceding byte count, whereas the in-memory representation does not. Character set is as indicated by the code page indicator.
VT_LPWSTR	A counted and null terminated Unicode string; a DWORD character count (where the count includes the terminating null) followed by that many Unicode (16bit) characters. Note that the count is a character count, not a byte count.
VT_FILETIME	64bit FILETIME structure as defined by Win32
VT_BLOB	A DWORD count of bytes, followed by that many bytes of data; the byte count does not include the four bytes for the length of the count itself: an empty blob would have a count of zero, followed by zero bytes. Thus, the serialized representation of a VT_BLOB is similar to that of a VT_BSTR but does not guarantee a null byte at the end of the data.
VT_STREAM	Indicates the value is stored in a stream which is sibling to the “Contents” stream. Following this type indicator is data in the format of a serialized VT_LPSTR which names the stream containing the data.
VT_STORAGE	Indicates the value is stored in an IStorage which is sibling to the “Contents” stream. Following this type indicator is data in the format of a serialized VT_LPSTR which names the IStorage containing the data.
VT_STREAMED_OBJECT	As in VT_STREAM but indicates that the stream contains a serialized object, which is a class id followed by initialization data for the class.
VT_STORED_OBJECT	As in VT_STORAGE but indicates that the designated IStorage contains a loadable object.

VT_BLOB_OBJECT	A BLOB containing a serialized object in the same representation as would appear in a VT_STREAMED_OBJECT. That is, following the VT_BLOB_OBJECT tag is a DWORD byte count of the remaining data (where the byte count does not include the size of itself) which is in the format of a class id followed by initialization data for that class. The only significant difference between VT_BLOB_OBJECT and VT_STREAMED_OBJECT is that the former does not have the system-level storage overhead that the latter would have, and is therefore more suitable for scenarios involving numbers of small objects.
VT_CF	A BLOB containing a clipboard format identifier followed by the data in that format. That is, following the VT_CF tag is data in the format of a VT_BLOB: a DWORD count of bytes, followed by that many bytes of data in the format of a packed VTCFREP described just below, followed immediately by an array of bytes as appropriate for data in the clipboard format format (text, metafile, or whatever).
VT_CLSID	A class ID (or other GUID).
VT_VECTOR	If the type indicator is one of the above values with this bit on in addition, then the value is a DWORD count of elements, followed by that many repetitions of the value. As an example, a type indicator of VT_LPSTR|VT_VECTOR has a DWORD element count, a DWORD byte count, the first string data, a DWORD byte count, the second string data, and so on.

Clipboard format identifiers, stored with the tag VT_CF, use one of five different representations:

```
typedef struct VTCFREP {
                        LONG   lTag;
                        BYTE   rgb[];
                        } VTCFREP;
```

The values for rgb are determined by the different values for lTag:

lTag Value	**rgb value**
-1L	a DWORD containing a built-in Windows clipboard format value.
-2L	a DWORD containing a Macintosh clipboard format value.
-3L	a GUID containging a format identifier (this is in little usage).
any positive value	a null-terminated string containing a Windows clipboard format name, one suitable for passing to RegisterClipboardFormat. The code page used for characters in the string is per the code page indicator. The “positive value” here is the length of the string, including the null byte at the end.
0L	no data (very rare usage)

As was mentioned above, all type/value pairs begin on a 32-bit boundary. It follows that in turn, the type indicators and values of a type value pair are so aligned. This means that values may be necessarily followed by null bytes to align a subsequent type/value pair.

However, *within* a vector of values, each repetition of a value is to be aligned with its *natural* alignment rather than with 32-bit alignment. In practice, this is only significant for types VT_I2 and VT_BOOL (which have 2-byte natural alignment); all other types have 4-byte natural alignment. Therefore, a value with type tag VT_I2 | VT_VECTOR would be

- a DWORD element count, followed by
- an sequence of packed 2-byte integers with *no* padding between them, whereas a value of with type tag VT_LPSTR | VT_VECTOR would be a DWORD element count, followed by
- a sequence of (DWORD cch, char rgch[]) strings, each of which may be followed by null padding to round to a 32-bit boundary.

A.3 ‘CompObj’ Stream Binary Format

A.3.0 Overview

The ‘CompObj’ stream in a storage object provides generic information regarding the native data contained in this storage object. This generic information is manipulated through the OLE API functions WriteFmtUserTypeStg and ReadFmtUserTypeStg and includes:

- User Type: a user readable string that indicates the type of the object.
- Clipboard Format: implies the names and structure of streams and sub-storages.

This document exposes the binary format of the data written by WriteFmtUserTypeStg and interpreted by ReadFmtUserTypeStg.

A.3.1 Format

The format consists of three basic parts, that represent versions of the stream written by different versions of the OLE2 libraries:

- Header, User Type (ANSI), Clipboard format (ANSI)
- ProgID (ANSI): optional, if not present, not Unicode information may follow
- Unicode versions of User Type, Clipboard format and ProgID: optional, if any Unicode information is present all three items have to be valid. Presence of the Unicode information is indicated by a “magic DWORD” value following the ANSI ProgID.

The following is a detailed description of the format using a pseudo C++ syntax where applicable.

A.3.1.1 Mandatory part

A.3.1.1.1 Stream name

```
// Stream name: L"\1CompObj"
```


A.3.1.1.2 Header

```
struct CompObjHdr// The leading data in the CompObj stream
{
DWORDdwVersionAndByteOrder;// First DWORD: LOWORD Ver
                                sion=0x0001, HIWORD=FFFE (ignored by
                                reader!)
DWORDdwFormat = 0x00000a03;      // OS Version: always Win 3.1
DWORDunused=-1L;          // Always a -1L in the stream

CLSIDclsidClass;        // Class ID of this object, identical
                                to the CLSID in the parent storage of
                                the stream
};
```

A.3.1.1.3 User Type

```
struct ANSIUserType
{
DWORDdwLenBytes;// length of User Type string in bytes
    including terminating 0
charszUserType[dwLenBytes];// User Type string (ANSI) terminated
                                with '\0'
}
```

A.3.1.1.4 Clipboard Format (ANSI)

```
LONGdwCFLen;// Length of clipboard format name
    // special values:
    // 0  no clipboard format
    // -1 DWORD with standard Windows CF
    //follows:
    //DWORD cfStdWin;
    // -2 DWORD with standard Apple Mac
    //intosh CF follows:
    //DWORD cfStdMac;
    // >0 Length in bytes of clipboard
    //format name including terminating 0
char szCFName[dwCFLen]; // Clipboard Format Name (ANSI) te
                          //  minated with '\0'
```

A.3.1.2 Optional: ProgID (ANSI)

The stream may end at this point. Versions of OLE before 2.01 provided only the data described in section 2.1.

If more data follows it is to be interpreted as follows:

```
struct ANSIProgID
{
DWORDdwLenBytes;// length of ProgID stream in bytes.
                // dwLenBytes<=40
charszProgID[dwLenBytes];// ProgID string (ANSI) terminated with '\0'
}
```

A.3.1.3 Optional: Unicode versions

Only if a ANSI ProgID was provided (possibly with ANSIProgID::dwLenBytes=0), the following data may follow:

A.3.1.3.1 Magic Number

```
DWORD dwMagicNumber =0x71B239F4;    // indicates Unicode UserType, CF
                                    // and ProgID follow (all three!)
```

A.3.1.3.2 User Type (Unicode)

```
struct UNICODEUserType
{
DWORD dwLenBytes;// Size of Unicode User
    Type in bytes (not cha
    acters!) including te
    minating 0.
WCHARwszUserType[dwLenBytes/sizeof(WCHAR)];// Unicode User Type
                                            //string,terminated with
                                            // '\0'.
};
```

A.3.1.3.3 Clipboard Format (Unicode)

```
LONGdwUnicodeCFLen;// Length of Unicode clipboard format
    name in bytes
    // special values:
    // 0 no clipboard format
    // -1 DWORD with standard Windows CF
    //follows:
    //DWORD cfStdWin;
    // -2 DWORD with standard Apple Mac
    //intosh CF follows:
    //DWORD cfStdMac;
    // >0 Length in bytes of clipboard
    //format name including terminating 0
WCHARszCFName[dwUnicodeCFLen/sizeof(WCHAR)]; // Clipboard Format
              //Name (Unicode) terminated with '\0'
```

A.3.1.3.4 ProgID (Unicode)

```
struct UNICODEProgID
{
DWORD dwLenBytes;// Size of Unicode ProgID in bytes (not characters!) including
                 // terminating '\'0.
WCHARwszProgID[dwLenBytes/sizeof(WCHAR)];// Unicode ProgID string, terminated
                                         // with '\'0.
};
```


References

1. CCIR Recommendation 709, Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange
2. Hunt, R.W.G. “Measuring Color”, Ellis Harwood Limited, Chichester, England, 1987
3. CIE Publication 15.2 “Colorimetry” second edition, CIE, Vienna, 1986
4. CCIR Recommendation 601–1, Encoding Parameters of Digital Television for Studios
5. ISO/TC 130/WG2 N438, International Color Profile format, March 26, 1995
6. *OLE 2 Programmers Reference, Volume One* Microsoft Press (1994)
7. Brockschmidt, Kraig. *Inside OLE2*, Microsoft Press (1994)
8. ICC Profile Format Specification version 3.2, International Color Consortium (1995)
9. ISO/IEC 10918-1 / ITU-T Recommendation T.81 “Information technology - Digital compression and coding of continuous-tone still images - Requirements and guidelines
10. *Computer graphics: principles and practice*, James D. Foley et al., 2nd ed. 1992, Addison-Wesley Systems programming series
11. *PostScript language reference manual*, Adobe Systems, Inc., 2nd ed., 1990, Addison-Wesley Publishing Company, Inc.
12. *Programmer’s Guide to the IVUE toolkit*, FITS Imaging, Copyright © 1993-1994, FITS Imaging
13. *The Unicode Standard*, The Unicode Consortium, 1991, Addison-Wesley.
14. Porter, T., and T. Duff. *Compositing digital images*, ACM Computer Graphics (SIGGRAPH), 1984, 18(3), 253-259, SIGGRAPH '84 Conference Proceedings.
15. Thompson, Kelvin. *Alpha Blending*, Graphic Gems. (1990) Academic Press, Inc. 210-211
16. Goldman, R. “Decomposing projective transformations,” *Graphics Gems 3*, 1992, Academic Press, pp 98-107.
17. Miller, Steven. *DEC/HP, Network Computing Architecture, Remote Procedure Call Run Time Extensions Specification*, Version OSF TX1.0.11, July 23, 1992, Appendix A “Universal Unique Indentifiers,” http://www.osf.org/dce.
18. Blinn, James F. *Jim Blinn’s Corner: Compositing Part 1: Theory*, IEEE Computer Graphics & Applications, September 1994, pp. 83-87. *Compositing Part 2: Practice*, IEEE Computer Graphics & Applications, November 1994, pp. 78-82.
19. “Storage naming conventions,” *OLE 2 Programmer’s reference*. Volume 1, Microsoft Press, 1995, pp. 596-596.
20. Kano, Nadine and A. Freytag. “The international character set conundrum: ANSI Unicode, and Microsoft Windows,” *Microsoft Systems Journal*, 1994 Volum 9, November 1994.
21. Photography - Electronic still picture cameras - determination of ISO speed (Working draft #6).
22. ISO 14524, Photography - Electronic still picture cameras - Methods for measuring the opto-electronic conversion functions (Working draft 4.0).

23. Programming Windows, Charles Petzold, 1990, Microsoft Press.
24. Inside Macintosh: Files, Addison-Wesley Publishing Co. (1992)

