Windows Metafile Format (wmf) Specification

Table of Contents

1 Introduction

This document is a specification of the Windows Metafile Format (WMF). A Windows **metafile**—also called a **vector image**—consists of a collection of records that can store an image. The stored image can be rendered by parsing and processing the metafile structure.

The Windows metafile begins with a header record, which may include the version of the metafile, its size, and the maximum number of objects that are defined. A metafile is "played back" when its records are converted to graphics commands and executed to render the image.

1.1 Glossary

The following terms are specific to this document:

Additive Color Model: A **color model**, which involves light emitted directly from a source or illuminant of some sort. The additive reproduction process usually uses red, green and blue light to produce the other colors.

Anisotropic: Refers to the properties of an image, such as the scaling of logical units to device units, which are not the same regardless of the direction (x-axis versus y-axis) that is measured. Contrast with **isotropic**.

Banding: A printing technique in which an application prints an image by dividing it into a number of bands and sending each band to the printer separately.

Bitmap: A collection of structures that contain a representation of a graphical image, a **logical palette**, dimensions and other information.

Commission Internationale de l'Eclairage International (CIE): An international Commission on Illumination in Vienna, Austria (www.cie.co.at) that sets standards for all aspects of lighting and illumination, including colorimetry, photometry and the measurement of visible and invisible radiation.

CIEXYZ: A widely-used, device-independent color standard developed by the **Commission Internationale de l'Éclairage (CIE)**. The **CIEXYZ** standard is based on color-matching experiments on human observers. No actual device is expected to produce colors in this **color space**. It is used as a means of converting colors from one **color space** to another. The primary colors in this **color space** are the abstract colors X, Y, and Z.

CMYK: A **color space** used for commercial printing and most color computer printers. In theory, cyan, magenta and yellow (CMY) can print all colors, but inks are not pure and black comes out muddy. The black (K) ink is required for quality black-and-white printing.

Color Correction: A process used in graphics image rendering and other disciplines to alter the overall quality of the light measured on a scale known as color temperature.

Color Matching: The conversion of a color, sent from its original **color space**, to its visually closest color in the destination **color space**. See also **Image Color Management (ICM)**.

Color Model: See **Color Space**.

Color Space: A system for describing color numerically. Also known as a "**color model**." Mathematically, a **color space** is a mapping of color components to a multidimensional coordinate system. The number of dimensions is generally two, three, or four. For example, if colors are expressed as a combination of the three components red, green, and blue, a three-dimensional space is sufficient to describe all possible colors. **Grayscales**, however, can be

mapped to a two-dimensional **color space**. If transparency is considered one of the components of a **red green blue (RGB)** color, four dimensions are appropriate.

Color Temperature: A characteristic of visible light is determined by comparing its chromaticity with a theoretical, heated black-body radiator. It is the temperature in degrees Kelvin at which the heated black-body radiator matches the color of a given light source.

Device Context: A structure that defines a set of graphic objects and their associated attributes, and the graphic modes that affect output. The graphic objects include a pen for line drawing, a brush for painting and filling, a **bitmap** for copying or scrolling parts of the screen, a **palette** for defining the set of available colors, a **region** for clipping and other operations, and a **path** for painting and drawing operations. All of these **device context** properties and objects together define an environment for graphics output.

Device-Independent Bitmap (DIB): A container for bitmapped graphics, which specifies characteristics of the **bitmap** such that it can be created using one application and loaded and displayed in another application, while retaining an identical appearance.

Dithering: A form of digital **halftoning**.

Encapsulated PostScript (EPS): A file of PostScript-language raw data that describes the appearance of a single page. **EPS** data can describe text, graphics, and images; but the primary purpose of an **EPS** file is to be encapsulated within another PostScript-language page definition.

Enhanced Metafile Format (EMF) : A file format that supports the device-independent definitions of images.

Font Association: The automatic pairing of a font that contains ideographs with a font that does not contain ideographs. **Font association** is used to maintain font attributes across changes in locale. For example, it allows the user to enter ideographic characters regardless of which font is selected.

Font Embedding: The process of attaching a font to a document so that the font may be used wherever the document is used, regardless of whether the font is installed on the system.

Font Mapper: An operating system component that maps specified font attributes to available, installed fonts on the system.

Gamma: The way brightness is distributed across the intensity spectrum by a graphics device. Depending on the device, the **gamma** may have a significant effect on the way colors are perceived. Technically, **gamma** is an expression of the relationship between input voltage and resulting output intensity. A perfect linear device would have a **gamma** of 1.0; a monitor or printer typically has a **gamma** in the range of 1.8 to 2.6, which effects midrange tones.

Gamma Correction: An adjustment to the light intensity (brightness) of an graphics device, in order to match the output more closely to the original image.

Graphics Device Interface (GDI): A Windows API, which is supported on 16-bit and 32-bit versions of Windows, for performing graphics operations and image manipulation on logical graphics objects.

Grayscale: A continuum of shades of gray used to represent an image. Continuous-tone images, such as black-and-white photographs, use an almost unlimited number of shades of gray. Conventional computer hardware and software, however, can only represent a limited gray, typically 16 or 256. Grayscaling is the process of converting a continuous-tone image to an image that a computer can manipulate.

Note that grayscaling is different from **dithering**. **Dithering** simulates shades of gray by altering the density and pattern of black and white dots. In grayscaling, each individual dot can have a different shade of gray.

Halftoning: The process of converting **grayscale** or continuous-tone graphics to a representation with a discrete number of gray or tone levels.

ICC Color Profile: An **International Color Consortium (ICC)**-approved color management standard for specifying the attributes of imaging devices such as scanners, digital cameras, monitors and printers so that the color of an image remains true from source to destination. A **color profile** can be embedded within the image itself.

Image Color Management (ICM): Color management technology that ensures that a color image, graphic, or text object is rendered as close as possible to its original intent on any **device**, despite differences in imaging technologies and color capabilities among devices.<1>

International Color Consortium (ICC): A group established in 1993 by eight industry vendors for the purpose of creating, promoting and encouraging the standardization and evolution of an open, vendor-neutral, cross-platform color management system architecture and components. The outcome of this co-operation was the development of the **ICC** profile specification. Version 4 of the specification, [ICC], is now widely used and has recently been approved as an International Standard, ISO 15076.

Isotropic: Refers to the properties of an image, such as the scaling of logical units to device units, which are the same regardless of the direction (x-axis versus y-axis) that is measured. Contrast with **anisotropic**.

Joint Photographic Experts Group (JPEG): A standard still-image format that is very popular due to its excellent compression capabilities. **JPEG** files are widely used for photographic images, but are not as well suited for compressing charts and diagrams, because text can become fuzzy. **JPEG** files use the **JPEG** File Interchange Format, as specified in [JFIF], and file extensions are .JPG or .JFF.

Logical Object: A graphics object that is defined with device-independent parameters, without assuming device specifics, such as color format or resolution.

Logical Palette: A **palette** that defines colors as device-independent values. Unlike the **system palette**, which has predefined, device-specific color definitions, a **logical palette** contains color values that can be defined entirely by an application. A **logical palette** entry must be mapped to the **system palette** entry in order for the custom colors to appear when the application is run.

Mapping Mode: The way in which logical (device-independent) coordinates are mapped to device-specific coordinates.

Metafile: A collection of structures that can store an image in an application-independent format. The stored image can be recreated by processing the **metafile** structures. Also called a **vector image**, a **metafile** contains a sequence of drawing commands, object definitions, and configuration settings. The commands, objects, and settings recorded in a **metafile** can be used to render its contents on a display, output by a printer or plotter, stored in memory, or saved to a file or stream.

METAFILEPICT: A structure that defines the **metafile** picture format. **METAFILEPICT** is used for exchanging **metafile** data through the clipboard. See [MSDN-CLIPFORM] for further information

N-UP Printing: Arranging multiple logical pages on a physical sheet of paper.

Packed Bitmap: A **Device-Independent Bitmap (DIB)** in which the bit array immediately follows a BitmapInfoHeader Object (section 2.2.1.2).

Palette: An array of values, each element of which contains the definition of a color. The color elements in a **palette** are often indexed so that clients can refer to the colors, each of which can occupy 24 bits or more, by a number that requires less storage space.

Path: A graphics object that is a container for a series of line and curve segments, and **regions** in an image.

Pitch: A property of a font that describes the horizontal density of characters in a font; that is, the number of characters that can fit in a given unit of space. When all the characters in a font have the same width, the font is called "fixed-pitch"; if characters can have various widths, the font is "variable-pitch".

Times New Roman is a variable-pitch font; it is easy to see that the characters in the font may have different widths. For example, the width of a lower-case "i" is visibly less than the width of an upper-case "W".

Playback Device Context: The **device_context** that defines the current graphics state during playback of the **metafile**. Although the data in an **WMF metafile** is device-independent, playback is always associated with an output device with specific properties, such as resolution, color support, etc.

Point: A unit of measurement used in printing for the font height, which is equal to 1/72 of an inch. Thus, a 12-point font defines a character cell approximately 1/6 of an inch tall. An 8½ × 11 inch form can generally fit 80 × 60 characters in a 12-point font.

Portable Network Graphics (PNG): A bitmapped graphics file format that provides advanced graphics features such as 48-bit color, alpha channels, built-in **gamma** and color correction, tight compression and the ability to display at one resolution and print at another.

Raster Operation: The process of combining the bits in a source **bitmap** with the bits in a destination **bitmap**.

Raw Mode: Refers to a spool file format that requires no further processing; it is ready to be received by the printer for which the data was formatted.

Red Green Blue (RGB): An **additive color model** in which red, green and blue are combined in various ways to reproduce other colors.

Region: A graphics object that is an area of an image, non-rectilinear in shape, that is defined by an array of scanlines.

Reverse Polish Notation: A mathematical notation wherein every operator follows all of its operands. Also known as Postfix notation.

Run-Length Encoding (RLE): The process of compressing a **bitmap** to reduce disk and memory space required for the **bitmap**. For more information, see section 3.1.5 .

sRGB: A standard, pre-defined **color space** that is portable across all devices and allows accurate **color matching** with little overhead. Developed by Hewlett-Packard and Microsoft, and specified in [IEC-RGB], **sRGB** is automatically available to users of Windows.<2>

Stock Object: A predefined graphics object. Stock objects are used as default brush, font, **palette** and pen objects in the **playback device context**.

System Palette: The **palette** that is actually in use to reproduce colors on a device such as a computer screen. A **system palette** has predefined, device-specific colors that are used by default, so that every application does not have to set them up.

Terminal Server: The computer on which nearly all of the computing resources reside that are used in a **Terminal Services** networking environment. The **terminal server** receives and processes keystrokes and mouse movements that take place on the client computer. The **terminal server** displays the desktop and running applications within a window on the client computer.

Terminal Services: A technology that allows multiple remote users to connect to a single server system. Clients of **Terminal Services** include Windows terminals and handheld PCs that are using Remote Desktop Protocol (RDP).

Tri-Stimulus: The generation of color using three color components.

TrueType: A scalable font technology that renders fonts for both the printer and the screen. Originally developed by Apple, it was enhanced jointly by Apple and Microsoft. Each **TrueType** font contains its own algorithms for converting printer outlines into screen bitmaps, which means both the outline and **bitmap** information is rasterized from the same font data. The lower-level language embedded within the **TrueType** font allows great flexibility in their design.

Typeface: A term that is used interchangeably with "font", however, a **typeface** is the primary design of a set of printed characters, such as Courier, Helvetica, and Times Roman, while a font is the particular implementation and variation of the **typeface**, such as normal, bold, or italic. The distinguishing characteristic of a **typeface** is often the presence or absence of serifs.

TWIP: A unit of measurement used in printing, equal to 1/20 **point**, or 1/1440 of an inch.

Vector Image: See **metafile**.

White Point: A set of **tri-stimulus** values that define the color "white" in graphics image rendering. Depending on the application, different definitions of white may needed to produce acceptable results. For example, photographs taken indoors may be lit by incandescent lights, which are relatively orange compared to daylight. Defining "white" as daylight will give unacceptable results when attempting to **color-correct** a photograph taken with incandescent lighting.

Windows Color System (WCS): Color management technology that ensures a color image, graphic or text object is rendered as closely as possible to its original intent on any device, despite differences in imaging technologies and color capabilities between devices. **WCS** is a superset of **ICM** APIs and functionality and includes a variety of new functions that provide significant improvements in flexiblity, transparency, predictability and extensibility for vendors. <3>

Windows Graphics Device Interface (GDI): See **Graphics Device Interface (GDI)**.

Windows Metafile Format (WMF): A file format used by Windows that supports the definition of images.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[ICC] International Color Consortium, "Image Technology Colour Management - Architecture, Profile Format, and Data Structure", Specification ICC.1:2004-10, May 2006, http://www.color.org/icc_specs2.xalter

[IEC-RGB] International Electrotechnical Commission, "Colour Measurement and Management in Multimedia Systems and Equipment - Part 2-1: Default RGB Colour Space - sRGB", May 1998, http://www.colour.org/tc8-05/Docs/colorspace/61966-2-1.pdf

[JFIF] Hamilton, E., "JPEG File Interchange Format, Version 1.02", September 1992, http://www.w3.org/Graphics/JPEG/jfif.txt

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[RFC2083] Boutell, T., et al., "PNG (Portable Network Graphics) Specification Version 1.0", RFC 2083, March 1997, http://www.ietf.org/rfc/rfc2083.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2781] Hoffman, P. and Yergeau, F., "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000, http://www.ietf.org/rfc/rfc2781.txt

[W3C-PNG] World Wide Web Consortium, "Portable Network Graphics (PNG) Specification, Second Edition", November 2003, http://www.w3.org/TR/PNG

1.2.2 Informative References

[MSDN-CLIPFORM] Microsoft Corporation, "Clipboard Formats", http://msdn2.microsoft.com/en-us/library/ms649013.aspx

[MS-EMF] Microsoft Corporation, "Enhanced Metafile Format Specification", July 2007.

[WGFX] Yuan, F., "Windows Graphics Programming - Win32 GDI and DirectDraw", Prentice Hall PTR, 2000, ISBN: 0130869856.

If you have any trouble finding [WGFX], please check here.

1.3 Structure Overview (Synopsis)

1.3.1 Metafile Structure

The Windows Metafile Format (WMF) specifies structures for defining a graphical image. A **WMF** metafile contains drawing commands, property definitions and graphics objects in a series of WMF records. In effect, a WMF metafile is a digital recording of an image, and the recording can be

played back to reproduce that image. Because WMF metafiles are application-independent, they can be shared among applications and used for image storage.

The following figure illustrates the high-level structure of a WMF metafile:

Figure 1: Structure of a Windows metafile

The WMF header record, specified in section 2.3.2.1, contains information that defines the characteristics of the metafile, including:

- The type of metafile
- The version of metafile
- The size of the metafile
- The number of objects defined in the metafile
- The size of the largest single record in the metafile

WMF records have a generic format, which is specified in section 2.3. Every WMF record contains the following information:

- The record size
- The record function
- Parameters, if any, for the record function

All WMF metafiles are terminated by a WMF end-of-file record.

1.3.2 Graphics Objects

Graphics objects, such as pens, brushes and bitmaps, which are used in the drawing and painting operations specified in the records of a Windows Metafile Format (WMF) metafile, may be defined in metafile records prior to the records that specify their use; that is, they are designed to be reusable during the course of processing the metafile.

Throughout this specification, it is assumed that these previously-defined, reusable graphics objects are available when needed for the processing of particular metafile records. This store of available objects is referred to in the text as the **WMF Object Table**, which is described in section 3.1.4.1 .

The exact characteristics of an object store for the WMF Objects are determined by the particular implementation that parses or writes the metafiles. Section 3.1.4 provides suggestions and examples concerning the management of these objects.

1.3.3 Byte Ordering

Data in the Windows Metafile Format (WMF) metafile records is stored in **little-endian** format.

Some computer architectures number bytes in a binary word from left to right, which is referred to as **big-endian**. The bit diagram for this documentation is big-endian. Other architectures number the bytes in a binary word from right to left, which is referred to as little-endian. The underlying file format enumerations, objects, and records are little-endian.

Using big-endian and little-endian methods, the number 0x12345678 would be stored as shown in the following:

Byte order	Byte 0	Byte 1	Byte 2	Byte 3
Big-endian	0x12	0x34	0x56	0x78
Little-endian	0x78	0x56	0x34	0x12

1.4 Relationship to Protocols and Other Records

The Windows Metafile Format (WMF) is not dependent on any protocols or other structures. The WMF defines a design and layout based on 16-bit operating systems. <4>

On 32-bit systems and for print spooling, it has been replaced by the Enhanced Metafile Format (EMF), defined in [MS-EMF].

1.5 Applicability Statement

Structures that are compliant with the Windows Metafile Format (WMF) are portable, application-independent containers for images. The graphics supported in WMF metafiles are applicable to document content representation, including printing and plotting.

1.6 Versioning and Localization

This document covers versioning issues in the following areas:

Structure Versions: There is only one version of the Windows Metafile Format structure.

Localization: This structure defines no locale-specific processes or data.

1.7 Vendor-Extensible Fields

The Windows Metafile Format (WMF) defines a mechanism for the encapsulation of arbitrary, vendor-defined data. See section 2.3.6.1 for details.

2 Structures

This section specifies the structures used to define the Windows Metafile Format, including:

- Enumerations of the Windows Metafile Format properties, styles and flags;
- Definitions of the Windows Metafile Format objects, in fixed and variable-length categories;
- Specifications of the Windows Metafile Format records, by type.

2.1 WMF Enumerations

This section contains enumerations of constant values that are referenced throughout the specification of Windows Metafile Format.

2.1.1 RecordType Enumeration

The Windows Metafile Format **RecordType Enumeration** defines every type of record that is found in a WMF metafile, except the **Header** record, which contains a Header object, specified in section 2.3.2.1 . The **Header** record does not need to be "recognized", since it MUST appear exactly once in the metafile, as the first record.

```
typedef  enum
{
  META_EOF = 0x0000,
  META_REALIZEPALETTE = 0x0035,
  META_SETPALENTRIES = 0x0037,
  META_SETBKMODE = 0x0102,
  META_SETMAPMODE = 0x0103,
  META_SETROP2 = 0x0104,
  META_SETRELABS = 0x0105,
  META_SETPOLYFILLMODE = 0x0106,
  META_SETSTRETCHBLTMODE = 0x0107,
  META_SETTEXTCHAREXTRA = 0x0108,
  META_RESTOREDC = 0x0127,
  META_RESIZEPALETTE = 0x0139,
  META_DIBCREATEPATTERNBRUSH = 0x0142,
  META_SETLAYOUT = 0x0149,
  META_SETBKCOLOR = 0x0201,
  META_SETTEXTCOLOR = 0x0209,
  META_OFFSETVIEWPORTORG = 0x0211,
  META_LINETO = 0x0213,
  META_MOVETO = 0x0214,
  META_OFFSETCLIPRGN = 0x0220,
  META_FILLREGION = 0x0228,
  META_SETMAPPERFLAGS = 0x0231,
  META_SELECTPALETTE = 0x0234,
  META_POLYGON = 0x0324,
  META_POLYLINE = 0x0325,
  META_SETTEXTJUSTIFICATION = 0x020A,
  META_SETWINDOWORG = 0x020B,
  META_SETWINDOWEXT = 0x020C,
  META_SETVIEWPORTORG = 0x020D,
  META_SETVIEWPORTEXT = 0x020E,
  META_OFFSETWINDOWORG = 0x020F,
  META_SCALEWINDOWEXT = 0x0410,
  META_SCALEVIEWPORTEXT = 0x0412,
```

```
  META_EXCLUDECLIPRECT = 0x0415,
  META_INTERSECTCLIPRECT = 0x0416,
  META_ELLIPSE = 0x0418,
  META_FLOODFILL = 0x0419,
  META_FRAMEREGION = 0x0429,
  META_ANIMATEPALETTE = 0x0436,
  META_TEXTOUT = 0x0521,
  META_POLYPOLYGON = 0x0538,
  META_EXTFLOODFILL = 0x0548,
  META_RECTANGLE = 0x041B,
  META_SETPIXEL = 0x041F,
  META_ROUNDRECT = 0x061C,
  META_PATBLT = 0x061D,
  META_SAVEDC = 0x001E,
  META_PIE = 0x081A,
  META_STRETCHBLT = 0x0B23,
  META_ESCAPE = 0x0626,
  META_INVERTREGION = 0x012A,
  META_PAINTREGION = 0x012B,
  META_SELECTCLIPREGION = 0x012C,
  META_SELECTOBJECT = 0x012D,
  META_SETTEXTALIGN = 0x012E,
  META_ARC = 0x0817,
  META_CHORD = 0x0830,
  META_BITBLT = 0x0922,
  META_EXTTEXTOUT = 0x0a32,
  META_SETDIBTODEV = 0x0d33,
  META_DIBBITBLT = 0x0940,
  META_DIBSTRETCHBLT = 0x0b41,
  META_STRETCHDIB = 0x0f43,
  META_DELETEOBJECT = 0x01f0,
  META_CREATEPALETTE = 0x00f7,
  META_CREATEPATTERNBRUSH = 0x01F9,
  META_CREATEPENINDIRECT = 0x02FA,
  META_CREATEFONTINDIRECT = 0x02FB,
  META_CREATEBRUSHINDIRECT = 0x02FC,
  META_CREATEREGION = 0x06FF
} RecordType;
```

META_EOF: This record specifies the end of the file, the last record in the metafile.

META_REALIZEPALETTE: This record maps **palette** entries from the current **logical palette** to the **system palette**.

META_SETPALENTRIES: This record defines **red green blue (RGB)** color values in a range of entries in a logical palette.

META_SETBKMODE: This record defines the background mix mode of the **playback device context**. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.

META_SETMAPMODE: This record defines the **mapping mode** of the playback device context. The mapping mode defines the unit of measure used to transform page-space units into device-space units, and also defines the orientation of the device's x and y axes.

META_SETROP2: This record defines the current foreground **raster operation** mixing mode. Metafile processing uses the foreground mix mode to combine pens and interiors of filled

objects with the colors already on the screen. The foreground mix mode defines how colors from the brush or pen and the colors in the existing image are to be combined.

META_SETRELABS: This record is reserved and not supported.

META_SETPOLYFILLMODE: This record defines the polygon fill mode for functions that fill polygons.

META_SETSTRETCHBLTMODE: This record defines the **bitmap** stretching mode in the playback device context.

META_SETTEXTCHAREXTRA: This record defines the intercharacter spacing. Intercharacter spacing is added to each character, including break characters, when the system writes a line of text.

META_RESTOREDC: This record defines a record that the playback device context SHOULD be set from a previously saved playback device context.

META_RESIZEPALETTE: This record redefines the size of a logical palette based on the specified value.

META_DIBCREATEPATTERNBRUSH: This record defines a brush with a pattern specified by a **DIB**.

META_SETLAYOUT: This record defines layout orientation options.

META_SETBKCOLOR: This record defines the current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.

META_SETTEXTCOLOR: This record defines the text color for the playback device context to the specified color.

META_OFFSETVIEWPORTORG: This record defines the viewport origin for a playback device context by using the specified horizontal and vertical offsets.

META_LINETO: This record defines a line from the current position up to, but not including, the specified point.

META_MOVETO: This record defines the current position to the specified point.

META_OFFSETCLIPRGN: This record defines the clipping **region** of a playback device context by the specified offsets.

META_FILLREGION: This record defines how to fill a region by using the specified brush.

META_SETMAPPERFLAGS: This record defines the algorithm that the **font mapper** uses when it maps logical fonts to physical fonts.

META_SELECTPALETTE: This record defines which palette to select as the current palette in a playback device context.

META_POLYGON: This record defines a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.

META_POLYLINE: This record defines a series of line segments by connecting the points in the specified array.

META_SETTEXTJUSTIFICATION: This record defines the amount of space the system SHOULD add to the break characters in a string of text.

META_SETWINDOWORG: This record defines which window point maps to the window origin (0,0).

META_SETWINDOWEXT: This record defines the horizontal and vertical extent of the window for a playback device context by using the specified values.

META_SETVIEWPORTORG: This record defines which device point maps to the viewport origin (0,0).

META_SETVIEWPORTEXT: This record defines the horizontal and vertical extent of the viewport for a playback device context by using the specified values.

META_OFFSETWINDOWORG: This record defines the window origin for a playback device context by using the specified horizontal and vertical offsets.

META_SCALEWINDOWEXT: This record defines the window for a playback device context by using the ratios formed by the specified multiplicands and divisors.

META_SCALEVIEWPORTEXT: This record defines the viewport for a playback device context by using the ratios formed by the specified multiplicands and divisors.

META_EXCLUDECLIPRECT: This record defines a clipping region that consists of the existing clipping region minus the specified rectangle.

META_INTERSECTCLIPRECT: This record defines a clipping region from the intersection of the current clipping region and the specified rectangle.

META_ELLIPSE: This record defines an ellipse. The center of the ellipse is the center of the specified bounding rectangle. The ellipse is outlined by using the current pen and is filled by using the current brush.

META_FLOODFILL: This record defines an area of the display surface with the current brush.

META_FRAMEREGION: This record defines a border around the specified region by using the specified brush.

META_ANIMATEPALETTE: This record redefines entries in the specified logical palette.

META_TEXTOUT: This record defines a character string at the specified location, by using the currently selected font, background color, and text color.

META_POLYPOLYGON: This record defines a series of closed polygons. Each polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode. The polygons drawn by this function can overlap.

META_EXTFLOODFILL: This record defines an area to be filled with the current brush.

META_RECTANGLE: This record defines a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.

META_SETPIXEL: This record sets the current pixel at the specified coordinates to the specified color.

META_ROUNDRECT: This record defines a rectangle with rounded corners. The rectangle is outlined by using the current pen and filled by using the current brush.

META_PATBLT: This record defines the data to paint the specified rectangle by using the brush that is currently selected into the playback device context. The brush color and the surface color or colors are combined by using the specified raster operation.

META_SAVEDC: This record defines a record that the playback device context SHOULD be saved for later retrieval.

META_PIE: This record defines a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The pie is outlined by using the current pen and filled by using the current brush.

META_STRETCHBLT: This record defines a bitmap from a source rectangle into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary. The bitmap is stretched or compressed according to the stretching mode currently set in the destination playback device context.

META_ESCAPE: This record is defined to access capabilities of a particular printing device that are not directly available through other well-defined Windows Metafile Format records in the **RecordType Enumeration** table.

META_INVERTREGION: This record defines the region in which the colors SHOULD be inverted.

META_PAINTREGION: This record defines how to paint the specified region by using the brush that is currently selected.

META_SELECTCLIPREGION: This record defines a region as the current clipping region for the playback device context.

META_SELECTOBJECT: This record defines the object to be selected as current into the playback device context. The new object replaces the previous object of the same type.

META_SETTEXTALIGN: This record defines the text-alignment values for the playback device context.

META_ARC: This record defines an elliptical arc.

META_CHORD: This record defines a chord, which is a region bounded by the intersection of an ellipse and a line segment. The chord is outlined by using the current pen and filled by using the current brush.

META_BITBLT: This record defines a bitmap corresponding to a rectangle of pixels from the specified source **device context** into a destination playback device context.

META_EXTTEXTOUT: This record specifies text to be output by using the currently selected font, background color, and text color. Optionally, dimensions can be provided for clipping, opaquing, or both.

META_SETDIBTODEV: This record defines the pixels in a specified rectangle by using color data from a **device-independent bitmap (DIB)**.

META_DIBBITBLT: This record defines a DIB based on the specified raster operations, source device context, and destination location.

META_DIBSTRETCHBLT: This record defines a DIB based on the specified raster operations, source device context, and destination location.

META_STRETCHDIB: This record defines a DIB based upon the color data for a rectangle of pixels in a DIB and the specified destination rectangle. If the destination rectangle is larger than the source rectangle, this function stretches the rows and columns of color data to fit the destination rectangle. If the destination rectangle is smaller than the source rectangle, this function compresses the rows and columns by using the specified raster operation.

META_DELETEOBJECT: This record deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object. After the object is deleted, the specified handle is no longer valid.

META_CREATEPALETTE: This record defines a palette.

META_CREATEPATTERNBRUSH: This record defines a brush with a pattern specified by a DIB.

META_CREATEPENINDIRECT: This record defines a pen that has the style, width, and color specified in a record.

META_CREATEFONTINDIRECT: This record defines a font that has the specified characteristics.

META_CREATEBRUSHINDIRECT: This record defines a brush that has the specified style, color, and pattern.

META_CREATEREGION: This record defines a region.

The high-order byte of the WMF record type values MAY<5> be ignored for all record types except the following.

- **META_BITBLT**
- **META_DIBBITBLT**
- **META_DIBSTRETCHBLT**
- **META_POLYGON**
- **META_POLYLINE**
- **META_SETPALENTRIES**
- **META_STRETCHBLT**

The meanings of the high-order bytes of these record types fields are specified in the respective sections that define them.

2.1.2 BinaryRasterOperation Enumeration

The Windows Metafile Format (WMF) **BinaryRasterOperation Enumeration** section lists the binary raster-operation codes. Raster-operation codes define how metafile processing combines the bits from the selected pen with the bits in the destination bitmap.

Each raster-operation code represents a Boolean operation in which the values of the pixels in the selected pen and the destination bitmap are combined. Following are the two operands used in these operations.

Operand	Meaning
P	Selected pen
D	Destination bitmap

Following are the Boolean operators used in these operations.

Operator	Meaning
a	Bitwise AND
n	Bitwise NOT (inverse)
o	Bitwise OR
x	Bitwise exclusive OR (XOR)

All Boolean operations are presented in **reverse Polish notation**. For example, the following operation replaces the values of the pixels in the destination bitmap with a combination of the pixel values of the pen and the selected brush: DPo.

Each raster-operation code is a 32-bit integer whose high-order word is a Boolean operation index and whose low-order word is the operation code. The 16-bit operation index is a zero-extended, 8-bit value that represents all possible outcomes resulting from the Boolean operation on two parameters (in this case, the pen and destination values). For example, the operation indexes for the DPo and DPan operations are shown in the following list.

P	D	DPo	DPan
0	0	0	1
0	1	1	1
1	0	1	1
1	1	1	0

The following enumeration lists the drawing modes and the Boolean operations that they represent.

```
typedef  enum
{
  R2_BLACK = 0x0001,
  R2_NOTMERGEPEN = 0x0002,
  R2_MASKNOTPEN = 0x0003,
  R2_NOTCOPYPEN = 0x0004,
  R2_MASKPENNOT = 0x0005,
  R2_NOT = 0x0006,
  R2_XORPEN = 0x0007,
  R2_NOTMASKPEN = 0x0008,
  R2_MASKPEN = 0x0009,
  R2_NOTXORPEN = 0x000A,
  R2_NOP = 0x000B,
  R2_MERGENOTPEN = 0x000C,
  R2_COPYPEN = 0x000D,
  R2_MERGEPENNOT = 0x000E,
```

```
  R2_MERGEPEN = 0x000F,
  R2_WHITE = 0x0010
} BinaryRasterOperation;
```

R2_BLACK: 0, Pixel is always 0.

R2_NOTMERGEPEN: DPon, Pixel is the inverse of the **R2_MERGEPEN** color.

R2_MASKNOTPEN: DPna, Pixel is a combination of the screen color and the inverse of the pen color.

R2_NOTCOPYPEN: Pn, Pixel is the inverse of the pen color.

R2_MASKPENNOT: PDna, Pixel is a combination of the colors common to both the pen and the inverse of the screen.

R2_NOT: Dn, Pixel is the inverse of the screen color.

R2_XORPEN: DPx, Pixel is a combination of the colors in the pen or in the screen, but not in both.

R2_NOTMASKPEN: DPan, Pixel is the inverse of the **R2_MASKPEN** color.

R2_MASKPEN: DPa, Pixel is a combination of the colors common to both the pen and the screen.

R2_NOTXORPEN: DPxn, Pixel is the inverse of the **R2_XORPEN** color.

R2_NOP: D, Pixel remains unchanged.

R2_MERGENOTPEN: DPno, Pixel is a combination of the colors common to both the screen and the inverse of the pen.

R2_COPYPEN: P, Pixel is the pen color.

R2_MERGEPENNOT: PDno, Pixel is a combination of the pen color and the inverse of the screen color.

R2_MERGEPEN: DPo, Pixel is a combination of the pen color and the screen color.

R2_WHITE: 1, Pixel is always 1

For a monochrome device, WMF format maps the value 0 to black and the value 1 to white. If an application attempts to draw with a black pen on a white destination by using the available binary raster operations, the following results occur.

Raster operation	Result
R2_BLACK	Visible black line
R2_COPYPEN	Visible black line
R2_MASKNOTPEN	No visible line
R2_MASKPEN	Visible black line
R2_MASKPENNOT	Visible black line

Raster operation	Result
R2_MERGENOTPEN	No visible line
R2_MERGEPEN	Visible black line
R2_MERGEPENNOT	Visible black line
R2_NOP	No visible line
R2_NOT	Visible black line
R2_NOTCOPYPEN	No visible line
R2_NOTMASKPEN	No visible line
R2_NOTMERGEPEN	Visible black line
R2_NOTXORPEN	Visible black line
R2_WHITE	No visible line
R2_XORPEN	No visible line

For a color device, WMF format uses RGB values to represent the colors of the pen and the destination. An RGB color value is a long integer that contains a red, a green, and a blue color field, each specifying the intensity of the given color. Intensities range from 0 through 255. The values are packed in the three low-order bytes of the long integer. The color of a pen is always a solid color, but the color of the destination may be a mixture of any two or three colors. If an application attempts to draw with a white pen on a blue destination by using the available binary raster operations, the following results occur.

Raster operation	Result
R2_BLACK	Visible black line
R2_COPYPEN	Visible white line
R2_MASKNOTPEN	Visible black line
R2_MASKPEN	Invisible blue line
R2_MASKPENNOT	Visible red/green line
R2_MERGENOTPEN	Invisible blue line
R2_MERGEPEN	Visible white line
R2_MERGEPENNOT	Visible white line
R2_NOP	Invisible blue line
R2_NOT	Visible red/green line
R2_NOTCOPYPEN	Visible black line
R2_NOTMASKPEN	Visible red/green line
R2_NOTMERGEPEN	Visible black line

Raster operation	Result
R2_NOTXORPEN	Invisible blue line
R2_WHITE	Visible white line
R2_XORPEN	Visible red/green line

2.1.3 BitCount Enumeration

The Windows Metafile Format **BitCount Enumeration** specifies the number of bits needed to define a pixel in a bitmap.

```
typedef  enum
{
  BI_BITCOUNT_0 = 0x0000,
  BI_BITCOUNT_1 = 0x0001,
  BI_BITCOUNT_2 = 0x0004,
  BI_BITCOUNT_3 = 0x0008,
  BI_BITCOUNT_4 = 0x0010,
  BI_BITCOUNT_5 = 0x0018,
  BI_BITCOUNT_6 = 0x0020
} BitCount;
```

BI_BITCOUNT_0: The bitmap bits-per-pixel is undefined; the **Compression Enumeration (section 2.1.8)** value in the BitmapInfoHeader Object (section 2.2.1.2) MUST be BI_JPEG or BI_PNG. There is no color table for **JPEG** or **PNG** images, so **BI_BITCOUNT_0** specifies that there is no color table for the bitmap.

The bits-per-pixel information for a JPEG or PNG image is encoded in the JPEG or PNG format; see [JFIF] and [RFC2083], respectively, for more information.

BI_BITCOUNT_1: The bitmap is monochrome, and the **Colors** field of a DIB object contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the **Colors** table; if the bit is set, the pixel has the color of the second entry in the table.

BI_BITCOUNT_2: The bitmap has a maximum of 16 colors, and the **Colors** field of the DIB contains up to 16 entries. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.

BI_BITCOUNT_3: The bitmap has a maximum of 256 colors, and the **Colors** field of the DIB contains up to 256 entries. In this case, each byte in the array represents a single pixel.

BI_BITCOUNT_4: The bitmap has a maximum of 2^16 colors.

If the **Compression** field of the BitmapInfoHeader Object is BI_RGB, the **Colors** field of DIB is NULL. Each **WORD** in the bitmap array represents a single pixel. The relative intensities of red, green, and blue are represented with five bits for each color component. The value for blue is in the least significant five bits, followed by five bits each for green and red. The most significant bit is not used. The **Colors** color table is used for optimizing colors used on palette-based devices, and MUST contain the number of entries specified by the **ColorUsed** field of the BitmapInfoHeader Object.

If the **Compression** field of the BitmapInfoHeader Object is set to BI_BITFIELDS, the **Colors** field contains three **DWORD** color masks that specify the red, green, and blue components, respectively, of each pixel. Each **WORD** in the bitmap array represents a single pixel.

When the **Compression** field is set to BI_BITFIELDS, bits set in each **DWORD** mask MUST be contiguous and SHOULD NOT overlap the bits of another mask.

BI_RGB and BI_BITFIELDS are defined in **Compression Enumeration**, section 2.1.8 .

BI_BITCOUNT_5: The bitmap has a maximum of 2^24 colors, and the **Colors** field of DIB is NULL. Each 3-byte triplet in the bitmap array represents the relative intensities of blue, green, and red, respectively, for a pixel. The **Colors**color table is used for optimizing colors used on palette-based devices, and MUST contain the number of entries specified by the **ColorUsed** field of the BitmapInfoHeader Object.

BI_BITCOUNT_6: The bitmap has a maximum of 2^24 colors.

If the **Compression** field of the BitmapInfoHeader Object is set to BI_RGB, the **Colors** field of DIB is set to NULL. Each **DWORD** in the bitmap array represents the relative intensities of blue, green, and red, respectively, for a pixel. The high byte in each **DWORD** is not used. The **Colors**color table is used for optimizing colors used on palette-based devices, and MUST contain the number of entries specified by the **ColorUsed** field of the BitmapInfoHeader Object.

If the **Compression** field of the BitmapInfoHeader Object is set to BI_BITFIELDS, the **Colors** field contains three **DWORD** color masks that specify the red, green, and blue components, respectively, of each pixel. Each **DWORD** in the bitmap array represents a single pixel.

When the **Compression** field is set to BI_BITFIELDS, bits set in each **DWORD** mask must be contiguous and should not overlap the bits of another mask. All the bits in the pixel do not need to be used.

BI_RGB and BI_BITFIELDS are specified in **Compression Enumeration**, section 2.1.8 .

2.1.4 BrushStyle Enumeration

The Windows Metafile Format (WMF) **BrushStyle Enumeration** specifies the different possible brush types that can be used in graphics operations. For more information, see the specification of the Brush Object (section 2.2.2.2).

```
typedef  enum
{
  BS_SOLID = 0x0000,
  BS_NULL = 0x0001,
  BS_HATCHED = 0x0002,
  BS_PATTERN = 0x0003,
  BS_INDEXED = 0x0004,
  BS_DIBPATTERN = 0x0005,
  BS_DIBPATTERNPT = 0x0006,
  BS_PATTERN8X8 = 0x0007,
  BS_DIBPATTERN8X8 = 0x0008,
  BS_MONOPATTERN = 0x0009
} BrushStyle;
```

BS_SOLID: A brush that paints a single, constant color, either solid or dithered.

BS_NULL: A brush that does nothing. Using a BS_NULL brush in a graphics operation MUST have the same effect as using no brush at all. <6>

BS_HATCHED: A brush that paints a predefined simple pattern, or "hatch", onto a solid background.

BS_PATTERN: A brush that paints a pattern defined by a bitmap, which MAY be a Bitmap16 Object or a DeviceIndependentBitmap (DIB) Object.

BS_INDEXED: Not supported.

BS_DIBPATTERN: A pattern brush specified by a DIB.

BS_DIBPATTERNPT: A pattern brush specified by a DIB.

BS_PATTERN8X8: Not supported.

BS_DIBPATTERN8X8: Not supported.

BS_MONOPATTERN: Not supported.

2.1.5 CharacterSet Enumeration

The Windows Metafile Format (WMF) **CharacterSet Enumeration** defines the superset of possible character sets that are available for fonts in WMF metafiles.

```
typedef  enum
{
  ANSI_CHARSET = 0x00000000,
  DEFAULT_CHARSET = 0x00000001,
  SYMBOL_CHARSET = 0x00000002,
  MAC_CHARSET = 0x0000004D,
  SHIFTJIS_CHARSET = 0x00000080,
  HANGUL_CHARSET = 0x00000081,
  JOHAB_CHARSET = 0x00000082,
  GB2312_CHARSET = 0x00000086,
  CHINESEBIG5_CHARSET = 0x00000088,
  GREEK_CHARSET = 0x000000A1,
  TURKISH_CHARSET = 0x000000A2,
  VIETNAMESE_CHARSET = 0x000000A3,
  HEBREW_CHARSET = 0x000000B1,
  ARABIC_CHARSET = 0x000000B2,
  BALTIC_CHARSET = 0x000000BA,
  RUSSIAN_CHARSET = 0x000000CC,
  THAI_CHARSET = 0x000000DE,
  EASTEUROPE_CHARSET = 0x000000EE,
  OEM_CHARSET = 0x000000FF
} CharacterSet;
```

ANSI_CHARSET: Specifies the English character set.

DEFAULT_CHARSET: Specifies a character set based on the current system locale; for example, when the system locale is United States English, the default character set is ANSI_CHARSET.

SYMBOL_CHARSET: Specifies a character set of symbols.

MAC_CHARSET: Specifies the Apple Macintosh character set. <7>

SHIFTJIS_CHARSET: Specifies the Japanese character set.

HANGUL_CHARSET: Also spelled "Hangeul", specifies the Hangul Korean character set.

JOHAB_CHARSET: Also spelled "Johap", specifies the Johab Korean character set.

GB2312_CHARSET: Specifies the "simplified" Chinese character set for People's Republic of China.

CHINESEBIG5_CHARSET: Specifies the "traditional" Chinese character set, used mostly in Taiwan, and in the Hong Kong and Macau Special Administrative Regions.

GREEK_CHARSET: Specifies the Greek character set.

TURKISH_CHARSET: Specifies the Turkish character set.

VIETNAMESE_CHARSET: Specifies the Vietnamese character set.

HEBREW_CHARSET: Hebrew character set

ARABIC_CHARSET: Arabic character set

BALTIC_CHARSET: Specifies the Baltic (Northeastern European) character set

RUSSIAN_CHARSET: Specifies the Russian Cyrillic character set.

THAI_CHARSET: Specifies the Thai character set.

EASTEUROPE_CHARSET: Specifies a Eastern European character set.

OEM_CHARSET: Specifies a mapping to one of the OEM code pages, according to the current system locale setting.

2.1.6 ClipPrecision Enumeration

The Windows Metafile Format **ClipPrecision Enumeration** specifies the clipping precision, which defines how to clip characters that are partially outside the clipping region. The values of this enumeration can be used individually or together, as the result of a bitwise OR.

```
typedef  enum
{
  CLIP_DEFAULT_PRECIS = 0x00000000,
  CLIP_CHARACTER_PRECIS = 0x00000001,
  CLIP_STROKE_PRECIS = 0x00000002,
  CLIP_LH_ANGLES = 0x00000010,
  CLIP_TT_ALWAYS = 0x00000020,
  CLIP_DFA_DISABLE = 0x00000040,
  CLIP_EMBEDDED = 0x00000080
} ClipPrecision;
```

CLIP_DEFAULT_PRECIS: Specifies that default clipping MUST be used.

CLIP_CHARACTER_PRECIS: This value SHOULD NOT <8> be used.

CLIP_STROKE_PRECIS: This value SHOULD NOT <9> be used.

CLIP_LH_ANGLES: This value MAY be used to control font rotation, as follows:

- If this bit value is clear (0), device fonts SHOULD rotate counterclockwise, but the rotation of other fonts SHOULD be determined by the orientation of the coordinate system.
- If this bit value is set (1), the rotation for all fonts SHOULD be determined by the orientation of the coordinate system; that is, whether the orientation is left-handed or right-handed.

CLIP_TT_ALWAYS: This value SHOULD NOT <10> be used.

CLIP_DFA_DISABLE: This value specifies that **font association** SHOULD <11> be turned off.

CLIP_EMBEDDED: This value specifies that **font embedding** MUST be used to render document content; embedded fonts MUST be read-only.

2.1.7 ColorUsage Enumeration

The Windows Metafile Format **ColorUsage Enumeration** specifies whether a color table exists and whether it contains explicit red, green, blue values or palette indices.

```
typedef  enum
{
  DIB_RGB_COLORS = 0x0000,
  DIB_PAL_COLORS = 0x0001,
  DIB_PAL_INDICES = 0x0002
} ColorUsage;
```

DIB_RGB_COLORS: The color table contains RGBQuad Object (section 2.2.1.15) values.

DIB_PAL_COLORS: The color table contains 16-bit indices into the device's color table.

DIB_PAL_INDICES: No color table exists. The bitmap contains indices into the device's color table.

2.1.8 Compression Enumeration

The Windows Metafile Format (WMF) **Compression Enumeration** specifies the type of compression for a compressed bitmap. Note that only bottom-up bitmaps can be compressed.

```
typedef  enum
{
  BI_RGB = 0x0000,
  BI_RLE8 = 0x0001,
  BI_RLE4 = 0x0002,
  BI_BITFIELDS = 0x0003,
  BI_JPEG = 0x0004,
  BI_PNG = 0x0005,
  BI_CMYK = 0x000B,
  BI_CMYKRLE8 = 0x000C,
  BI_CMYKRLE4 = 0x000D
} Compression;
```

BI_RGB: An uncompressed format.

BI_RLE8: A red green blue (RGB) **run-length encoded (RLE)** format for bitmaps with 8 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by a byte containing a color index. For more information, see Bitmap Compression (section 3.1.5).

BI_RLE4: An RGB RLE format for bitmaps with 4 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes. For more information, see Bitmap Compression.

BI_BITFIELDS: The bitmap is not compressed and the color table consists of three **DWORD** color masks that specify the red, green, and blue components, respectively, of each pixel. This is valid when used with 16 and 32-bits per pixel bitmaps.

BI_JPEG: The image is a Joint Photographic Experts Group (JPEG) image, as specified in [JFIF]. This value SHOULD only be used in certain bitmap operations, such as JPEG pass-through. The application MUST query for the pass-through support, since not all devices support JPEG pass-through. Using non-RGB bitmaps MAY limit the portability of the metafile to other devices. For instance, display device contexts generally do not support this pass-through.

BI_PNG: The image is a Portable Network Graphics (PNG) image, as specified in [RFC2083]. This value SHOULD only be used certain bitmap operations, such as JPEG/PNG pass-through. The application MUST query for the pass-through support, since not all devices support JPEG/PNG pass-through. Using non-RGB bitmaps MAY limit the portability of the metafile to other devices. For instance, display device contexts generally do not support this pass-through.

BI_CMYK: The image is an uncompressed **CMYK** format.

BI_CMYKRLE8: A CMYK RLE format for bitmaps with 8 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by a byte containing a color index.

BI_CMYKRLE4: A CMYK RLE format for bitmaps with 4 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes.

2.1.9 ExtTextOutOptions Enumeration

The Windows Metafile Format **ExtTextOutOptions Enumeration** specifies parameters that control various aspects of the output of text by META_EXTTEXTOUT (section 2.3.3.5) records

```
typedef  enum
{
  ETO_OPAQUE = 0x0002,
  ETO_CLIPPED = 0x0004,
  ETO_GLYPH_INDEX = 0x0010,
  ETO_RTLREADING = 0x0080,
  ETO_NUMERICSLOCAL = 0x0400,
  ETO_NUMERICSLATIN = 0x0800,
  ETO_IGNORELANGUAGE = 0x1000,
  ETO_PDY = 0x2000
} ExtTextOutOptions;
```

ETO_OPAQUE: This value indicates that the current background color SHOULD be used to fill the rectangle.

ETO_CLIPPED: This value indicates that the text SHOULD be clipped to the rectangle.

ETO_GLYPH_INDEX: This value indicates that the string to be output SHOULD NOT require further processing with respect to the placement of the characters, and an array of character placement values SHOULD be provided. This character placement process is useful for fonts in which diacritical characters affect character spacing.

ETO_RTLREADING: This value indicates that the text MUST be laid out in right-to-left reading order, instead of the default left-to-right order. This SHOULD be applied only when the font selected into the playback device context is either Hebrew or Arabic.

ETO_NUMERICSLOCAL: This value indicates that to display numbers, digits appropriate to the locale SHOULD be used.

ETO_NUMERICSLATIN: This value indicates that to display numbers, European digits SHOULD be used.

ETO_IGNORELANGUAGE: This value is reserved and SHOULD NOT be used.

ETO_PDY: This value indicates that both horizontal and vertical character displacement values SHOULD be provided.

2.1.10 FamilyFont Enumeration

The Windows Metafile Format (WMF) **FamilyFont Enumeration** specifies the font family. Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact **typeface** desired is not available.

```
typedef  enum
{
  FF_DONTCARE = 0x00,
  FF_ROMAN = 0x01,
  FF_SWISS = 0x02,
  FF_MODERN = 0x03,
  FF_SCRIPT = 0x04,
  FF_DECORATIVE = 0x05
} FamilyFont;
```

FF_DONTCARE: Use default font.

FF_ROMAN: Fonts with variable stroke width (proportional) and with serifs. MS Serif is an example.

FF_SWISS: Fonts with variable stroke width (proportional) and without serifs. MS Sans Serif is an example.

FF_MODERN: Fonts with constant stroke width (monospace), with or without serifs. Monospace fonts are usually modern. Pica, Elite, and Courier New are examples.

FF_SCRIPT: Fonts designed to look like handwriting. Script and Cursive are examples.

FF_DECORATIVE: Novelty fonts. Old English is an example.

When the **FamilyFont Enumeration** value is packed in a byte with a **PitchFont Enumeration** value, the combination is called "PitchAndFamily". Bits 0-1 specify the **pitch** and bits 4-7 specify the family.

2.1.11 FloodFill Enumeration

The Windows Metafile Format **FloodFill Enumeration** specifies the type of fill operation to be performed.

```
typedef  enum
{
  FLOODFILLBORDER = 0x0000,
  FLOODFILLSURFACE = 0x0001
} FloodFill;
```

FLOODFILLBORDER: The fill area is bounded by the color specified by the **Color** member. This style is identical to the filling performed by the META_FLOODFILL record.

FLOODFILLSURFACE: The fill area is defined by the color that is specified by **Color**. Filling continues outward in all directions as long as the color is encountered. This style is useful for filling areas with multicolored boundaries.

2.1.12 FontQuality Enumeration

The Windows Metafile Format **FontQuality Enumeration** specifies the output quality, which defines how closely the attributes of the logical font should match those of the actual physical font.

```
typedef  enum
{
  DEFAULT_QUALITY = 0x00,
  DRAFT_QUALITY = 0x01,
  PROOF_QUALITY = 0x02,
  NONANTIALIASED_QUALITY = 0x03,
  ANTIALIASED_QUALITY = 0x04,
  CLEARTYPE_QUALITY = 0x05
} FontQuality;
```

DEFAULT_QUALITY: Appearance of the font does not matter, so **DRAFT_QUALITY** MAY be used.

DRAFT_QUALITY: Appearance of the font is less important than when **PROOF_QUALITY** is used. For raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts MAY be synthesized if necessary.

PROOF_QUALITY: Character quality of the font SHOULD be given more weight than exact matching of the logical font attributes. For raster fonts, scaling SHOULD be disabled, and the font closest in size SHOULD be chosen. Although the chosen font size MAY NOT be mapped exactly when **PROOF_QUALITY** is used, the quality of the font MUST be high with no distortion of appearance. Bold, italic, underline, and strikeout fonts MAY be synthesized if necessary.

NONANTIALIASED_QUALITY: Font MUST NOT be antialiased; otherwise, **DEFAULT_QUALITY** MAY be used.

ANTIALIASED_QUALITY: Font MUST be antialiased, if the font supports it and the size of the font is not too small or too large.

CLEARTYPE_QUALITY: If set, text SHOULD be rendered using **ClearType** antialiasing.

2.1.13 GamutMappingIntent Enumeration

The Windows Metafile Format **GamutMappingIntent Enumeration** specifies the relationship between logical and physical colors.

```
typedef  enum
{
  LCS_GM_ABS_COLORIMETRIC = 0x00000008,
  LCS_GM_BUSINESS = 0x00000001,
  LCS_GM_GRAPHICS = 0x00000002,
  LCS_GM_IMAGES = 0x00000004
} GamutMappingIntent;
```

LCS_GM_ABS_COLORIMETRIC: Specifies that the **white point** SHOULD be maintained. Typically used when logical colors MUST be matched to their nearest physical color in the destination **color gamut**.

Intent: Match

ICC name: Absolute Colorimetric

LCS_GM_BUSINESS: Specifies that saturation SHOULD be maintained. Typically used for business charts and other situations in which **dithering** is not required.

Intent: Graphic

ICC name: Saturation

LCS_GM_GRAPHICS: Specifies that a colorimetric match SHOULD be maintained. Typically used for graphic designs and named colors.

Intent: Proof

ICC name: Relative Colorimetric

LCS_GM_IMAGES: Specifies that contrast SHOULD be maintained. Typically used for photographs and natural images.

Intent: Picture

ICC name: Perceptual

2.1.14 HatchStyle Enumeration

The Windows Metafile Format **HatchStyle Enumeration** specifies the hatch pattern.

```
typedef  enum
{
  HS_HORIZONTAL = 0x0000,
  HS_VERTICAL = 0x0001,
  HS_FDIAGONAL = 0x0002,
  HS_BDIAGONAL = 0x0003,
  HS_CROSS = 0x0004,
  HS_DIAGCROSS = 0x0005
```

```
} HatchStyle;
```

HS_HORIZONTAL: A horizontal hatch.

HS_VERTICAL: A vertical hatch.

HS_FDIAGONAL: A 45-degree downward, left-to-right hatch.

HS_BDIAGONAL: A 45-degree upward, left-to-right hatch.

HS_CROSS: A horizontal and vertical cross-hatch.

HS_DIAGCROSS: A 45-degree crosshatch.

2.1.15 Layout Enumeration

The Windows Metafile Format **Layout Enumeration** specifies the layout, which is the order in which text and graphics are revealed.

```
typedef  enum
{
  LAYOUT_RTL = 0x00000001,
  LAYOUT_BTT = 0x00000002,
  LAYOUT_VBH = 0x00000004,
  LAYOUT_BITMAPORIENTATIONPRESERVED = 0x00000008
} Layout;
```

LAYOUT_RTL: Sets the default horizontal layout to be right-to-left.

LAYOUT_BTT: Sets the default horizontal layout to be bottom-to-top.

LAYOUT_VBH: Sets the default layout to be vertical before horizontal.

LAYOUT_BITMAPORIENTATIONPRESERVED: Disables any reflection during META_BITBLT and META_STRETCHBLT operations.

2.1.16 LogicalColorSpace Enumeration

The Windows Metafile Format (WMF) **LogicalColorSpace Enumeration** specifies the type of **color space**.

```
typedef  enum
{
  LCS_CALIBRATED_RGB = 0x00000000,
  LCS_sRGB = 0x73524742,
  LCS_WINDOWS_COLOR_SPACE = 0x57696E20
} LogicalColorSpace;
```

LCS_CALIBRATED_RGB: Color values are calibrated red green blue (RGB) values. The values are translated by using the endpoints specified by the **Endpoints** member before being passed to the device.

LCS_sRGB: The value is an encoding of the **ASCII** characters "**sRGB**", and it indicates that the color values are sRGB values.

LCS_WINDOWS_COLOR_SPACE: The value is an encoding of the ASCII characters "Win ", including the trailing space, and it indicates that the color values are Windows default color space values.

2.1.17 LogicalColorSpaceV5 Enumeration

The Windows Metafile Format (WMF) **LogicalColorSpaceV5 Enumeration** is used to specify where to find **color profile** information for a DeviceIndependentBitmap (DIB) Object (section 2.2.2.3) that has a header of type BitmapV5Header Object (section 2.2.1.4).

```
typedef  enum
{
  LCS_PROFILE_LINKED = 0x4C494E4B,
  LCS_PROFILE_EMBEDDED = 0x4D424544
} LogicalColorSpaceV5;
```

LCS_PROFILE_LINKED: The value consists of the string "LINK" from the Windows character set (code page 1252). It indicates that the color profile MUST be linked with the DIB Object.

LCS_PROFILE_EMBEDDED: The value consists of the string "MBED" from the Windows character set (code page 1252). It indicates that the color profile MUST be embedded in the DIB Object.

2.1.18 MapMode Enumeration

The Windows Metafile Format **MapMode Enumeration** defines how logical units are mapped to physical units; that is, assuming that the origins in both the logical and physical coordinate systems are at the same point on the drawing surface, what is the physical coordinate (x',y') that corresponds to logical coordinate (x,y).

For example, suppose the mapping mode is **MM_TEXT**. Given the definition of that mapping mode below, and an origin (0,0) at the top left corner of the drawing surface, logical coordinate (4,5) would map to physical coordinate (4,5) in pixels.

Now suppose the mapping mode is **MM_LOENGLISH**, with the same origin as the previous example. Given the definition of the mapping mode below, logical coordinate (4,-5) would map to physical coordinate (0.04,0.05) in inches.

```
typedef  enum
{
  MM_TEXT = 0x0001,
  MM_LOMETRIC = 0x0002,
  MM_HIMETRIC = 0x0003,
  MM_LOENGLISH = 0x0004,
  MM_HIENGLISH = 0x0005,
  MM_TWIPS = 0x0006,
  MM_ISOTROPIC = 0x0007,
  MM_ANISOTROPIC = 0x0008
} MapMode;
```

MM_TEXT: Each logical unit is mapped to one device pixel. Positive x is to the right; positive y is down.

MM_LOMETRIC: Each logical unit is mapped to 0.1 millimeter. Positive x is to the right; positive y is up.

MM_HIMETRIC: Each logical unit is mapped to 0.01 millimeter. Positive x is to the right; positive y is up.

MM_LOENGLISH: Each logical unit is mapped to 0.01 inch. Positive x is to the right; positive y is up.

MM_HIENGLISH: Each logical unit is mapped to 0.001 inch. Positive x is to the right; positive y is up.

MM_TWIPS: Each logical unit is mapped to one twentieth (1/20) of a point. In printing, a point is 1/72 of an inch; therefore, 1/20 of a point is 1/1440 of an inch. This unit is also known as a "**twip**".

Positive x is to the right; positive y is up.

MM_ISOTROPIC: Logical units are mapped to arbitrary device units with equally-scaled axes; that is, one unit along the x-axis is equal to one unit along the y-axis. The META_SETWINDOWEXT and META_SETVIEWPORTEXT records specify the units and the orientation of the axes.

The processing application SHOULD make adjustments as necessary to ensure the x and y units remain the same size. For example, when the window extent is set, the viewport SHOULD be adjusted to keep the units **isotropic**.

MM_ANISOTROPIC: Logical units are mapped to arbitrary units with arbitrarily-scaled axes.

2.1.19 MetafileEscapes Enumeration

The Windows Metafile Format **MetafileEscapes Enumeration** specifies extensions to WMF functionality that are not directly available through other records defined in the WMF **RecordType Enumeration** table in section 2.1.1 . Many of these functions specify output operations on a printer, or the configuration of a printer, by interfacing with a **printer driver**.

Before using any of these extended functions in a metafile, an implementation MUST first call the **QUERY_ESCSUPPORT** escape to find out if the escape is supported or not.

```
typedef  enum
{
  NEWFRAME = 0x0001,
  ABORTDOC = 0x0002,
  NEXTBAND = 0x0003,
  SET_COLORTABLE = 0x0004,
  GET_COLORTABLE = 0x0005,
  FLUSHOUT = 0x0006,
  DRAFTMODE = 0x0007,
  QUERY_ESCSUPPORT = 0x0008,
  SET_ABORTPROC = 0x0009,
  STARTDOC = 0x000A,
  ENDDOC = 0x000B,
  GET_PHYSPAGESIZE = 0x000C,
  GET_PRINTINGOFFSET = 0x000D,
```

```
  GET_SCALINGFACTOR = 0x000E,
  MFCOMMENT = 0x000F,
  SET_PENWIDTH = 0x0010,
  SET_COPYCOUNT = 0x0011,
  SET_PAPERSOURCE = 0x0012,
  PASSTHROUGH = 0x0013,
  GET_TECHNOLOGY = 0x0014,
  SET_LINECAP = 0x0015,
  SET_LINEJOIN = 0x0016,
  SET_MITERLIMIT = 0x0017,
  BANDINFO = 0x0018,
  DRAWPATTERNRECT = 0x0019,
  GET_VECTORPENSIZE = 0x001A,
  GET_VECTORBRUSHSIZE = 0x001B,
  ENABLEDUPLEX = 0x001C,
  GETSET_PAPERBINS = 0x001D,
  GETSET_PRINTORIENT = 0x001E,
  ENUM_PAPERBINS = 0x001F,
  SET_DIBSCALING = 0x0020,
  EPS_PRINTING = 0x0021,
  ENUM_PAPERMETRICS = 0x0022,
  GETSET_PAPERMETRICS = 0x0023,
  POSTSCRIPT_DATA = 0x0025,
  POSTSCRIPT_IGNORE = 0x0026,
  GET_DEVICEUNITS = 0x002A,
  GET_EXTENDED_TEXTMETRICS = 0x0100,
  GET_PAIRKERNTABLE = 0x0102,
  EXTTEXTOUT = 0x0200,
  GET_FACENAME = 0x0201,
  DOWNLOAD_FACE = 0x0202,
  METAFILE_DRIVER = 0x0801,
  QUERY_DIBSUPPORT = 0x0C01,
  BEGIN_PATH = 0x1000,
  CLIP_TO_PATH = 0x1001,
  END_PATH = 0x1002,
  POSTSCRIPT_IDENTIFY = 0x1005,
  POSTSCRIPT_INJECTION = 0x1006,
  CHECK_JPEGFORMAT = 0x1007,
  CHECK_PNGFORMAT = 0x1008,
  GET_PS_FEATURESETTING = 0x1009,
  TS_QUERYVER = 0x100A,
  TS_RECORD = 0x100B,
  OPEN_CHANNEL = 0x1010,
  DOWNLOAD_HEADER = 0x1011,
  CLOSE_CHANNEL = 0x1012,
  POSTSCRIPT_PASSTHROUGH = 0x1013,
  ENCAPSULATED_POSTSCRIPT = 0x1014,
  SPCLPASSTHROUGH2 = 0x11D8
} MetafileEscapes;
```

NEWFRAME: Notifies the printer driver that the application has finished writing to a page.

ABORTDOC: Stops processing the current document.

NEXTBAND: Notifies the printer driver that the application has finished writing to a band.

<12>

SET_COLORTABLE: Sets color table values.

GET_COLORTABLE: Gets color table values.

FLUSHOUT: Causes all pending output to be flushed to the output device.

DRAFTMODE: Indicates that the printer driver should print text only, and no graphics.

QUERY_ESCSUPPORT: Queries a printer driver to determine whether a specific escape function is supported on the output device it drives.

SET_ABORTPROC: Sets the application-defined function that allows a **print job** to be canceled during printing.

STARTDOC: Notifies the printer driver that a new print job is starting.

ENDDOC: Notifies the printer driver that a new print job is ending.

GET_PHYSPAGESIZE: Retrieves the physical page size currently selected on an output device.

GET_PRINTINGOFFSET: Retrieves the offset from the upper-left corner of the physical page where the actual printing or drawing begins.

GET_SCALINGFACTOR: Retrieves the scaling factors for the x-axis and the y-axis of a printer.

MFCOMMENT: Adds arbitrary private data to the metafile. This data MAY be completely application-specific and undefined in this specification.

SET_PENWIDTH: Sets the width of a pen in pixels.

SET_COPYCOUNT: Sets the number of copies.

SET_PAPERSOURCE: Sets the source, such as a particular paper tray or bin on a printer, for output forms.

PASSTHROUGH: This record passes through arbitrary data.

GET_TECHNOLOGY: Gets information concerning graphics technology that is supported on an device.

SET_LINECAP: Specifies the line-drawing mode to use in output to a device.

SET_LINEJOIN: Specifies the line-joining mode to use in output to a device.

SET_MITERLIMIT: Sets the limit for the length of miter joins to use in output to a device.

BANDINFO: Retrieves or specifies settings concerning **banding** on a device, such as the number of bands.

DRAWPATTERNRECT: Draws a rectangle with a defined pattern.

GET_VECTORPENSIZE: Retrieves the physical pen size currently defined on a device.

GET_VECTORBRUSHSIZE: Retrieves the physical brush size currently defined on a device.

ENABLEDUPLEX: Enables or disables double-sided (duplex) printing on a device.

GETSET_PAPERBINS: Retrieves or specifies the source of output forms on a device.

GETSET_PRINTORIENT: Retrieves or specifies the paper orientation on a device.

ENUM_PAPERBINS: Retrieves information concerning the sources of different forms on an output device.

SET_DIBSCALING: Specifies the scaling of device-independent bitmaps (DIBs).

EPS_PRINTING: Indicates the start and end of an **encapsulated PostScript (EPS)** section.

ENUM_PAPERMETRICS: Query a printer driver for paper dimensions and other forms data.

GETSET_PAPERMETRICS: Retrieve or specify paper dimensions and other forms data on an output device.

POSTSCRIPT_DATA: Sends arbitrary **PostScript** data to an output device.

POSTSCRIPT_IGNORE: Notifies an output device to ignore PostScript data.

GET_DEVICEUNITS: Gets the device units currently configured on an output device.

GET_EXTENDED_TEXTMETRICS: Gets extended text metrics currently configured on an output device.

GET_PAIRKERNTABLE: Gets the font kern table currently defined on an output device.

EXTTEXTOUT: Draws text using the currently selected font, background color, and text color.

GET_FACENAME: Gets the font face name currently configured on a device.

DOWNLOAD_FACE: Sets the font face name on a device.

METAFILE_DRIVER: Queries a printer driver about the support for metafiles on an output device.

QUERY_DIBSUPPORT: Queries the printer driver about its support for DIBs on an output device.

BEGIN_PATH: Opens a **path**.

CLIP_TO_PATH: Defines a clip region that is bounded by a path. The input MUST be a 16-bit quantity that defines the action to take.

END_PATH: Ends a path.

POSTSCRIPT_IDENTIFY: Sets the printer driver to either PostScript-centric or **GDI**-centric mode.

POSTSCRIPT_INJECTION: Inserts a block of raw data into a PostScript stream. The input MUST be a 32-bit quantity specifying the number of bytes to inject, a 16-bit quantity specifying the injection point, and a 16-bit quantity specifying the page number, followed by the bytes to inject.

CHECK_JPEGFORMAT: Queries the printer driver to determine whether or not it can handle the given JPEG image.

CHECK_PNGFORMAT: Queries the printer driver to determine whether or not it can handle the given PNG image.

GET_PS_FEATURESETTING: Queries the printer driver for information about PostScript features supported on the output device.

TS_QUERYVER: Queries a display driver for **Terminal Services** support.

TS_RECORD: Sends a **Terminal Server** record to a display device.

OPEN_CHANNEL: The same as **STARTDOC**, with a NULL document and output filename, and data in **raw mode**.

DOWNLOAD_HEADER: Instructs the printer driver to download sets of PostScript procedures.

CLOSE_CHANNEL: Same as ENDDOC. See OPEN_CHANNEL.

POSTSCRIPT_PASSTHROUGH: Sends arbitrary data directly to a printer driver, which is expected to process this data only when in PostScript mode. See POSTSCRIPT_IDENTIFY.

ENCAPSULATED_POSTSCRIPT: Sends arbitrary data directly to the printer driver.

SPCLPASSTHROUGH2: Enables documents to include private procedures and other arbitrary data in documents.

2.1.20 MetafileType Enumeration

The Windows Metafile Format **MetafileType Enumeration** specifies where the metafile is stored.

```
typedef  enum
{
  MEMORYMETAFILE = 0x0001,
  DISKMETAFILE = 0x0002
} MetafileType;
```

MEMORYMETAFILE: Metafile is stored in memory.

DISKMETAFILE: Metafile is stored on disk.

2.1.21 MetafileVersion Enumeration

The Windows Metafile Format **MetafileVersion Enumeration** defines the version of WMF in terms of support for device-independent bitmaps (DIBs).

```
typedef  enum
{
  METAVERSION100 = 0x0100,
  METAVERSION300 = 0x0300
} MetafileVersion;
```

METAVERSION100: DIBs are not supported.

METAVERSION300: DIBs are supported.

2.1.22 MixMode Enumeration

The Windows Metafile Format **MixMode Enumeration** specifies the background mix mode for text, hatched brushes, and other non-solid pen styles.

```
typedef  enum
{
  TRANSPARENT = 0x0001,
  OPAQUE = 0x0002
} MixMode;
```

TRANSPARENT: The background remains untouched.

OPAQUE: The background is filled with the current background color before the text, hatched brush, or pen is drawn.

2.1.23 OutPrecision Enumeration

The Windows Metafile Format **OutPrecision Enumeration** specifies the output precision, which defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type.

```
typedef  enum
{
  OUT_DEFAULT_PRECIS = 0x00000000,
  OUT_STRING_PRECIS = 0x00000001,
  OUT_CHARACTER_PRECIS = 0x00000002,
  OUT_STROKE_PRECIS = 0x00000003,
  OUT_TT_PRECIS = 0x00000004,
  OUT_DEVICE_PRECIS = 0x00000005,
  OUT_RASTER_PRECIS = 0x00000006,
  OUT_TT_ONLY_PRECIS = 0x00000007,
  OUT_OUTLINE_PRECIS = 0x00000008,
  OUT_SCREEN_OUTLINE_PRECIS = 0x00000009,
  OUT_PS_ONLY_PRECIS = 0x0000000A
} OutPrecision;
```

OUT_DEFAULT_PRECIS: This value specifies the default font mapper behavior.

OUT_STRING_PRECIS: This value is not used by the font mapper, but it is returned when raster fonts are enumerated.

OUT_CHARACTER_PRECIS: Not used.

OUT_STROKE_PRECIS: This value is not used by the font mapper, but it is returned when **TrueType**, other outline-based fonts, and vector fonts are enumerated.

OUT_TT_PRECIS: This value specifies that the font mapper is to choose a TrueType font when the system contains multiple fonts with the same name.

OUT_DEVICE_PRECIS: This value specifies the font mapper to choose a device font when the system contains multiple fonts with the same name.

OUT_RASTER_PRECIS: This value specifies that the font mapper is to choose a raster font when the system contains multiple fonts with the same name.

OUT_TT_ONLY_PRECIS: This value specifies that the font mapper is to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.

OUT_OUTLINE_PRECIS: This value specifies that the font mapper is to choose from TrueType and other outline-based fonts.

OUT_SCREEN_OUTLINE_PRECIS: This value specifies that the font mapper is to have a preference for TrueType and other outline-based fonts.

OUT_PS_ONLY_PRECIS: This value specifies that the font mapper is to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.

2.1.24 PaletteEntryFlag Enumeration

The Windows Metafile Format **PaletteEntryFlag Enumeration** specifies how the palette entry should be used.

```
typedef  enum
{
  PC_RESERVED = 0x01,
  PC_EXPLICIT = 0x02,
  PC_NOCOLLAPSE = 0x04
} PaletteEntryFlag;
```

PC_RESERVED: Specifies that the logical palette entry be used for palette animation. This value prevents other windows from matching colors to the palette entry since the color frequently changes. If an unused system-palette entry is available, the color is placed in that entry. Otherwise, the color is not available for animation.

PC_EXPLICIT: Specifies that the low-order word of the logical palette entry designates a hardware palette index. This value allows the application to show the contents of the display device palette.

PC_NOCOLLAPSE: Specifies that the color be placed in an unused entry in the system palette instead of being matched to an existing color in the system palette. If there are no unused entries in the system palette, the color is matched normally. Once this color is in the system palette, colors in other logical palettes can be matched to this color.

2.1.25 PenStyle Enumeration

The 16-bit Windows Metafile Format **PenStyle Enumeration** is used to specify different types of pens that can be used in graphics operations.

Various styles can be combined by using a logical OR statement, one from each subsection of Style, EndCap, Join, and Type (Cosmetic).

```
typedef  enum
{
  PS_COSMETIC = 0x0000,
  PS_ENDCAP_ROUND = 0x0000,
  PS_JOIN_ROUND = 0x0000,
  PS_SOLID = 0x0000,
  PS_DASH = 0x0001,
```

```
  PS_DOT = 0x0002,
  PS_DASHDOT = 0x0003,
  PS_DASHDOTDOT = 0x0004,
  PS_NULL = 0x0005,
  PS_INSIDEFRAME = 0x0006,
  PS_USERSTYLE = 0x0007,
  PS_ALTERNATE = 0x0008,
  PS_ENDCAP_SQUARE = 0x0100,
  PS_ENDCAP_FLAT = 0x0200,
  PS_JOIN_BEVEL = 0x1000,
  PS_JOIN_MITER = 0x2000
} PenStyle;
```

PS_COSMETIC: The pen is cosmetic.

PS_ENDCAP_ROUND: Line end caps are round.

PS_JOIN_ROUND: Line joins are round.

PS_SOLID: The pen is solid.

PS_DASH: The pen is dashed.

PS_DOT: The pen is dotted.

PS_DASHDOT: The pen has alternating dashes and dots.

PS_DASHDOTDOT: The pen has dashes and double dots.

PS_NULL: The pen is invisible.

PS_INSIDEFRAME: The pen is solid. When this pen is used in any drawing record that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen.

PS_USERSTYLE: The pen uses a styling array supplied by the user.

PS_ALTERNATE: The pen sets every other pixel (this style is applicable only for cosmetic pens).

PS_ENDCAP_SQUARE: Line end caps are square.

PS_ENDCAP_FLAT: Line end caps are flat.

PS_JOIN_BEVEL: Line joins are beveled.

PS_JOIN_MITER: Line joins are mitered when they are within the current limit set by the SETMITERLIMIT META_ESCAPE record. A join is beveled when it would exceed the limit.

2.1.26 PitchFont Enumeration

The Windows Metafile Format **PitchFont Enumeration** defines values that are used for specifying characteristics of a font. The values are used to indicate whether the characters in a font have a fixed or variable width, or pitch.

```
typedef  enum
{
```

```
  DEFAULT_PITCH = 0,
  FIXED_PITCH = 1,
  VARIABLE_PITCH = 2
} PitchFont;
```

DEFAULT_PITCH: The default pitch is specified.

FIXED_PITCH: The font has a fixed pitch, so all the characters occupy the same width when output.

VARIABLE_PITCH: The font has a variable pitch, so the characters have widths that are proportional to the actual widths of the glyphs. Thus, "i" and the space character have much smaller width than "W" or "O".

When the **PitchFont** value is packed in a byte with a **FamilyFont Enumeration** value, the combination is called "PitchAndFamily". Bits 0-1 specify the pitch and bits 4-7 specify the family.

2.1.27 PolyFillMode Enumeration

The Windows Metafile Format **PolyFillMode Enumeration** specifies the method used for filling a polygon.

```
typedef  enum
{
  ALTERNATE = 0x0001,
  WINDING = 0x0002
} PolyFillMode;
```

ALTERNATE: Selects alternate mode (fills the area between odd-numbered and even-numbered polygon sides on each scan line).

WINDING: Selects winding mode (fills any region with a non-zero winding value).

2.1.28 PostScriptCap Enumeration

The Windows Metafile Format **PostScriptCap Enumeration** defines line-ending types for use with a PostScript printer driver.

```
typedef  enum
{
  PostScriptGdiCap = -1,
  PostscriptFlatCap = 0,
  PostScriptRoundCap = 1,
  PostScriptSquareCap = 2
} PostScriptCap;
```

PostScriptGdiCap: This value SHOULD NOT be used.

PostscriptFlatCap: Specifies that the line ends at the last point. The end is squared off.

PostScriptRoundCap: Specifies a circular cap. The center of the circle is the last point in the line. The diameter of the circle is the same as the line width; that is, the thickness of the line.

PostScriptSquareCap: Specifies a square cap. The center of the square is the last point in the line. The height and width of the square are the same as the line width; that is, the thickness of the line.

2.1.29 PostScriptFeatureSetting Enumeration

The Windows Metafile Format **PostScriptFeatureSetting Enumeration** defines values that correspond to specific features in a PostScript printer driver.

```
typedef  enum
{
  FEATURESETTING_NUP = 0,
  FEATURESETTING_OUTPUT = 1,
  FEATURESETTING_PSLEVEL = 2,
  FEATURESETTING_CUSTPAPER = 3,
  FEATURESETTING_MIRROR = 4,
  FEATURESETTING_NEGATIVE = 5,
  FEATURESETTING_PROTOCOL = 6
} PostScriptFeatureSetting;
```

FEATURESETTING_NUP: Refers to the page layout property for **n-up printing**.

FEATURESETTING_OUTPUT: Refers to the driver output options property.

FEATURESETTING_PSLEVEL: Refers to the PostScript language level.

FEATURESETTING_CUSTPAPER: Refers to the custom paper parameters property.

FEATURESETTING_MIRROR: Refers to the mirrored output property.

FEATURESETTING_NEGATIVE: Refers to the negative output property.

FEATURESETTING_PROTOCOL: Refers to the output protocol property.

2.1.30 PostScriptJoin Enumeration

The Windows Metafile Format **PostScriptJoin Enumeration** defines line-joining capabilities for use with a PostScript printer driver.

```
typedef  enum
{
  PostScriptNotSet = -2,
  PostScriptGdiJoin = -1,
  PostScriptMiterJoin = 0,
  PostScriptRoundJoin = 1,
  PostScriptBevelJoin = 2
} PostScriptJoin;
```

PostScriptNotSet: Specifies that the line-joining style has not been set, and that a default style MAY <13> be used.

PostScriptGdiJoin: This value SHOULD NOT be used.

PostScriptMiterJoin: Specifies a mitered join. This value MUST produce a sharp or clipped corner.

PostScriptRoundJoin: Specifies a circular join. This value MUST produce a smooth, circular arc between the lines.

PostScriptBevelJoin: Specifies a beveled join. This value MUST produce a diagonal corner.

2.1.31 StretchMode Enumeration

The Windows Metafile Format **StretchMode Enumeration** specifies the bitmap stretching mode, which defines how the system combines rows or columns of a bitmap with existing pixels.

```
typedef  enum
{
  BLACKONWHITE = 0x0001,
  WHITEONBLACK = 0x0002,
  COLORONCOLOR = 0x0003,
  HALFTONE = 0x0004
} StretchMode;
```

BLACKONWHITE: Performs a Boolean AND operation by using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves black pixels at the expense of white pixels.<14>

WHITEONBLACK: Performs a Boolean OR operation by using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves white pixels at the expense of black pixels.<15>

COLORONCOLOR: Deletes the pixels. This mode deletes all eliminated lines of pixels without trying to preserve their information.<16>

HALFTONE: Maps pixels from the source rectangle into blocks of pixels in the destination rectangle. The average color over the destination block of pixels approximates the color of the source pixels.<17>

After setting the **HALFTONE** stretching mode, the brush origin should be set to avoid misalignment artifacts.

2.1.32 TernaryRasterOperation Enumeration

The Windows Metafile Format (WMF) **TernaryRasterOperation Enumeration** specifies ternary raster operation codes, which define how to combine the bits in a source bitmap with the bits in a destination bitmap.

```
typedef  enum
{
  BLACKNESS = 0x00,
  DPSOON = 0x01,
  DPSONA = 0x02,
  PSON = 0x03,
  SDPONA = 0x04,
  DPON = 0x05,
  PDSXNON = 0x06,
  PDSAON = 0x07,
  SDPNAA = 0x08,
  PDSXON = 0x09,
  DPNA = 0x0A,
```

```
PSDNAON = 0x0B,
SPNA = 0x0C,
PDSNAON = 0x0D,
PDSONON = 0x0E,
PN = 0x0F,
PDSONA = 0x10,
NOTSRCERASE = 0x11,
SDPXNON = 0x12,
SDPAON = 0x13,
DPSXNON = 0x14,
DPSAON = 0x15,
PSDPSANAXX = 0x16,
SSPXDSXAXN = 0x17,
SPXPDXA = 0x18,
SDPSANAXN = 0x19,
PDSPAOX = 0x1A,
SDPSXAXN = 0x1B,
PSDPAOX = 0x1C,
DSPDXAXN = 0x1D,
PDSOX = 0x1E,
PDSOAN = 0x1F,
DPSNAA = 0x20,
SDPXON = 0x21,
DSNA = 0x22,
SPDNAON = 0x23,
SPXDSXA = 0x24,
PDSPANAXN = 0x25,
SDPSAOX = 0x26,
SDPSXNOX = 0x27,
DPSXA = 0x28,
PSDPSAOXXN = 0x29,
DPSANA = 0x2A,
SSPXPDXAXN = 0x2B,
SPDSOAX = 0x2C,
PSDNOX = 0x2D,
PSDPXOX = 0x2E,
PSDnoan = 0x2F,
PSNA = 0x30,
SDPNAON = 0x31,
SDPSOOX = 0x32,
NOTSRCCOPY = 0x33,
SPDSAOX = 0x34,
SPDSXNOX = 0x35,
SDPOX = 0x36,
SDPOAN = 0x37,
PSDPOAX = 0x38,
SPDNOX = 0x39,
SPDSXOX = 0x3A,
SPDNOAN = 0x3B,
PSX = 0x3C,
SPDSONOX = 0x3D,
SPDSNAOX = 0x3E,
PSAN = 0x3F,
PSDNAA = 0x40,
DPSXON = 0x41,
SDXPDXA = 0x42,
SPDSANAXN = 0x43,
SRCERASE = 0x44,
```

```
DPSNAON = 0x45,
DSPDAOX = 0x46,
PSDPXAXN = 0x47,
SDPXA = 0x48,
PDSPDAOXXN = 0x49,
DPSDOAX = 0x4A,
PDSNOX = 0x4B,
SDPANA = 0x4C,
SSPXDSXOXN = 0x4D,
PDSPXOX = 0x4E,
PDSNOAN = 0x4F,
PDNA = 0x50,
DSPNAON = 0x51,
DPSDAOX = 0x52,
SPDSXAXN = 0x53,
DPSONON = 0x54,
DSTINVERT = 0x55,
DPSOX = 0x56,
DPSOAN = 0x57,
PDSPOAX = 0x58,
DPSNOX = 0x59,
PATINVERT = 0x5A,
DPSDONOX = 0x5B,
DPSDXOX = 0x5C,
DPSNOAN = 0x5D,
DPSDNAOX = 0x5E,
DPAN = 0x5F,
PDSXA = 0x60,
DSPDSAOXXN = 0x61,
DSPDOAX = 0x62,
SDPNOX = 0x63,
SDPSOAX = 0x64,
DSPNOX = 0x65,
SRCINVERT = 0x66,
SDPSONOX = 0x67,
DSPDSONOXXN = 0x68,
PDSXXN = 0x69,
DPSAX = 0x6A,
PSDPSOAXXN = 0x6B,
SDPAX = 0x6C,
PDSPDOAXXN = 0x6D,
SDPSNOAX = 0x6E,
PDXNAN = 0x6F,
PDSANA = 0x70,
SSDXPDXAXN = 0x71,
SDPSXOX = 0x72,
SDPNOAN = 0x73,
DSPDXOX = 0x74,
DSPNOAN = 0x75,
SDPSNAOX = 0x76,
DSAN = 0x77,
PDSAX = 0x78,
DSPDSOAXXN = 0x79,
DPSDNOAX = 0x7A,
SDPXNAN = 0x7B,
SPDSNOAX = 0x7C,
DPSXNAN = 0x7D,
SPXDSXO = 0x7E,
```

```
DPSAAN = 0x7F,
DPSAA = 0x80,
SPXDSXON = 0x81,
DPSXNA = 0x82,
SPDSNOAXN = 0x83,
SDPXNA = 0x84,
PDSPNOAXN = 0x85,
DSPDSOAXX = 0x86,
PDSAXN = 0x87,
SRCAND = 0x88,
SDPSNAOXN = 0x89,
DSPNOA = 0x8A,
DSPDXOXN = 0x8B,
SDPNOA = 0x8C,
SDPSXOXN = 0x8D,
SSDXPDXAX = 0x8E,
PDSANAN = 0x8F,
PDSXNA = 0x90,
SDPSNOAXN = 0x91,
DPSDPOAXX = 0x92,
SPDAXN = 0x93,
PSDPSOAXX = 0x94,
DPSAXN = 0x95,
DPSXX = 0x96,
PSDPSONOXX = 0x97,
SDPSONOXN = 0x98,
DSXN = 0x99,
DPSNAX = 0x9A,
SDPSOAXN = 0x9B,
SPDNAX = 0x9C,
DSPDOAXN = 0x9D,
DSPDSAOXX = 0x9E,
PDSXAN = 0x9F,
DPA = 0xA0,
PDSPNAOXN = 0xA1,
DPSNOA = 0xA2,
DPSDXOXN = 0xA3,
PDSPONOXN = 0xA4,
PDXN = 0xA5,
DSPNAX = 0xA6,
PDSPOAXN = 0xA7,
DPSOA = 0xA8,
DPSOXN = 0xA9,
D = 0xAA,
DPSONO = 0xAB,
SPDSXAX = 0xAC,
DPSDAOXN = 0xAD,
DSPNAO = 0xAE,
DPNO = 0xAF,
PDSNOA = 0xB0,
PDSPXOXN = 0xB1,
SSPXDSXOX = 0xB2,
SDPANAN = 0xB3,
PSDNAX = 0xB4,
DPSDOAXN = 0xB5,
DPSDPAOXX = 0xB6,
SDPXAN = 0xB7,
PSDPXAX = 0xB8,
```

```
DSPDAOXN = 0xB9,
DPSNAO = 0xBA,
MERGEPAINT = 0xBB,
SPDSANAX = 0xBC,
SDXPDXAN = 0xBD,
DPSXO = 0xBE,
DPSANO = 0xBF,
MERGECOPY = 0xC0,
SPDSNAOXN = 0xC1,
SPDSONOXN = 0xC2,
PSXN = 0xC3,
SPDNOA = 0xC4,
SPDSXOXN = 0xC5,
SDPNAX = 0xC6,
PSDPOAXN = 0xC7,
SDPOA = 0xC8,
SPDOXN = 0xC9,
DPSDXAX = 0xCA,
SPDSAOXN = 0xCB,
SRCCOPY = 0xCC,
SDPONO = 0xCD,
SDPNAO = 0xCE,
SPNO = 0xCF,
PSDNOA = 0xD0,
PSDPXOXN = 0xD1,
PDSNAX = 0xD2,
SPDSOAXN = 0xD3,
SSPXPDXAX = 0xD4,
DPSANAN = 0xD5,
PSDPSAOXX = 0xD6,
DPSXAN = 0xD7,
PDSPXAX = 0xD8,
SDPSAOXN = 0xD9,
DPSDANAX = 0xDA,
SPXDSXAN = 0xDB,
SPDNAO = 0xDC,
SDNO = 0xDD,
SDPXO = 0xDE,
SDPANO = 0xDF,
PDSOA = 0xE0,
PDSOXN = 0xE1,
DSPDXAX = 0xE2,
PSDPAOXN = 0xE3,
SDPSXAX = 0xE4,
PDSPAOXN = 0xE5,
SDPSANAX = 0xE6,
SPXPDXAN = 0xE7,
SSPXDSXAX = 0xE8,
DSPDSANAXXN = 0xE9,
DPSAO = 0xEA,
DPSXNO = 0xEB,
SDPAO = 0xEC,
SDPXNO = 0xED,
SRCPAINT = 0xEE,
SDPNOO = 0xEF,
PATCOPY = 0xF0,
PDSONO = 0xF1,
PDSNAO = 0xF2,
```

```
  PSNO = 0xF3,
  PSDNAO = 0xF4,
  PDNO = 0xF5,
  PDSXO = 0xF6,
  PDSANO = 0xF7,
  PDSAO = 0xF8,
  PDSXNO = 0xF9,
  DPO = 0xFA,
  PATPAINT = 0xFB,
  PSO = 0xFC,
  PSDNOO = 0xFD,
  DPSOO = 0xFE,
  WHITENESS = 0xFF
} TernaryRasterOperation;
```

BLACKNESS:

Reverse Polish = 00000042

Common = 0

DPSOON:

Reverse Polish = 00010289

Common = DPSoon

DPSONA:

Reverse Polish = 00020C89

Common = DPSona

PSON:

Reverse Polish = 000300AA

Common = PSon

SDPONA:

Reverse Polish = 00040C88

Common = SDPona

DPON:

Reverse Polish = 000500A9

Common = DPon

PDSXNON:

Reverse Polish = 00060865

Common = PDSxnon

PDSAON:

Reverse Polish = 000702C5

Common = PDSaon

SDPNAA:

Reverse Polish = 00080F08

Common = SDPnaa

PDSXON:

Reverse Polish = 00090245

Common = PDSxon

DPNA:

Reverse Polish = 000A0329

Common = DPna

PSDNAON:

Reverse Polish = 000B0B2A

Common = PSDnaon

SPNA:

Reverse Polish = 000C0324

Common = SPna

PDSNAON:

Reverse Polish = 000D0B25

Common = PDSnaon

PDSONON:

Reverse Polish = 000E08A5

Common = PDSonon

PN:

Reverse Polish = 000F0001

Common = Pn

PDSONA:

Reverse Polish = 00100C85

Common = PDSona

NOTSRCERASE:

Reverse Polish = 001100A6

Common = DSon

SDPXNON:

Reverse Polish = 00120868

Common = SDPxnon

SDPAON:

Reverse Polish = 001302C8

Common = SDPaon

DPSXNON:

Reverse Polish = 00140869

Common = DPSxnon

DPSAON:

Reverse Polish = 001502C9

Common = DPSaon

PSDPSANAXX:

Reverse Polish = 00165CCA

Common = PSDPSanaxx

SSPXDSXAXN:

Reverse Polish = 00171D54

Common = SSPxDSxaxn

SPXPDXA:

Reverse Polish = 00180D59

Common = SPxPDxa

SDPSANAXN:

Reverse Polish = 00191CC8

Common = SDPSanaxn

PDSPAOX:

Reverse Polish = 001A06C5

Common = PDSPaox

SDPSXAXN:

Reverse Polish = 001B0768

Common = SDPSxaxn

PSDPAOX:

Reverse Polish = 001C06CA

Common = PSDPaox

DSPDXAXN:

Reverse Polish = 001D0766

Common = DSPDxaxn

PDSOX:

Reverse Polish = 001E01A5

Common = PDSox

PDSOAN:

Reverse Polish = 001F0385

Common = PDSoan

DPSNAA:

Reverse Polish = 00200F09

Common = DPSnaa

SDPXON:

Reverse Polish = 00210248

Common = SDPxon

DSNA:

Reverse Polish = 00220326

Common = DSna

SPDNAON:

Reverse Polish = 00230B24

Common = SPDnaon

SPXDSXA:

Reverse Polish = 00240D55

Common = SPxDSxa

PDSPANAXN:

Reverse Polish = 00251CC5

Common = PDSPanaxn

SDPSAOX:

Reverse Polish = 002606C8

Common = SDPSaox

SDPSXNOX:

Reverse Polish = 00271868

Common = SDPSxnox

DPSXA:

Reverse Polish = 00280369

Common = DPSxa

PSDPSAOXXN:

Reverse Polish = 002916CA

Common = PSDPSaoxxn

DPSANA:

Reverse Polish = 002A0CC9

Common = DPSana

SSPXPDXAXN:

Reverse Polish = 002B1D58

Common = SSPxPDxaxn

SPDSOAX:

Reverse Polish = 002C0784

Common = SPDSoax

PSDNOX:

Reverse Polish = 002D060A

Common = PSDnox

PSDPXOX:

Reverse Polish = 002E064A

Common = PSDPxox

PSDnoan:

Reverse Polish = 002F0E2A

Common = PSDnoan

PSNA:

Reverse Polish = 0030032A

Common = PSna

SDPNAON:

Reverse Polish = 00310B28

Common = SDPnaon

SDPSOOX:

Reverse Polish = 00320688

Common = SDPSoox

NOTSRCCOPY:

Reverse Polish = 00330008

Common = Sn

SPDSAOX:

Reverse Polish = 003406C4

Common = SPDSaox

SPDSXNOX:

Reverse Polish = 00351864

Common = SPDSxnox

SDPOX:

Reverse Polish = 003601A8

Common = SDPox

SDPOAN:

Reverse Polish = 00370388

Common = SDPoan

PSDPOAX:

Reverse Polish = 0038078A

Common = PSDPoax

SPDNOX:

Reverse Polish = 0390604

Common = SPDnox

SPDSXOX:

Reverse Polish = 003A0644

Common = SPDSxox

SPDNOAN:

Reverse Polish = 003B0E24

Common = SPDnoan

PSX:

Reverse Polish = 003C004A

Common = PSx

SPDSONOX:

Reverse Polish = 003D18A4

Common = SPDSonox

SPDSNAOX:

Reverse Polish = 003E1B24

Common = SPDSnaox

PSAN:

Reverse Polish = 003F00EA

Common = PSan

PSDNAA:

Reverse Polish = 00400F0A

Common = PSDnaa

DPSXON:

Reverse Polish = 00410249

Common = DPSxon

SDXPDXA:

Reverse Polish = 00420D5D

Common = SDxPDxa

SPDSANAXN:

Reverse Polish = 00431CC4

Common = SPDSanaxn

SRCERASE:

Reverse Polish = 00440328

Common = SDna

DPSNAON:

Reverse Polish = 00450B29

Common = DPSnaon

DSPDAOX:

Reverse Polish = 004606C6

Common = DSPDaox

PSDPXAXN:

Reverse Polish = 0047076A

Common = PSDPxaxn

SDPXA:

Reverse Polish = 00480368

Common = SDPxa

PDSPDAOXXN:

Reverse Polish = 004916C5

Common = PDSPDaoxxn

DPSDOAX:

Reverse Polish = 004A0789

Common = DPSDoax

PDSNOX:

Reverse Polish = 004B0605

Common = PDSnox

SDPANA:

Reverse Polish = 004C0CC8

Common = SDPana

SSPXDSXOXN:

Reverse Polish = 004D1954

Common = SSPxDSxoxn

PDSPXOX:

Reverse Polish = 004E0645

Common = PDSPxox

PDSNOAN:

Reverse Polish = 004F0E25

Common = PDSnoan

PDNA:

Reverse Polish = 00500325

Common = PDna

DSPNAON:

Reverse Polish = 00510B26

Common = DSPnaon

DPSDAOX:

Reverse Polish = 005206C9

Common = DPSDaox

SPDSXAXN:

Reverse Polish = 00530764

Common = SPDSxaxn

DPSONON:

Reverse Polish = 005408A9

Common = DPSonon

DSTINVERT:

Reverse Polish = 00550009

Common = Dn

DPSOX:

Reverse Polish = 005601A9

Common = DPSox

DPSOAN:

Reverse Polish = 000570389

Common = DPSoan

PDSPOAX:

Reverse Polish = 00580785

Common = PDSPoax

DPSNOX:

Reverse Polish = 00590609

Common = DPSnox

PATINVERT:

Reverse Polish = 005A0049

Common = DPx

DPSDONOX:

Reverse Polish = 005B18A9

Common = DPSDonox

DPSDXOX:

Reverse Polish = 005C0649

Common = DPSDxox

DPSNOAN:

Reverse Polish = 005D0E29

Common = DPSnoan

DPSDNAOX:

Reverse Polish = 005E1B29

Common = DPSDnaox

DPAN:

Reverse Polish = 005F00E9

Common = DPan

PDSXA:

Reverse Polish = 00600365

Common = PDSxa

DSPDSAOXXN:

Reverse Polish = 006116C6

Common = DSPDSaoxxn

DSPDOAX:

Reverse Polish = 00620786

Common = DSPDoax

SDPNOX:

Reverse Polish = 00630608

Common = SDPnox

SDPSOAX:

Reverse Polish = 00640788

Common = SDPSoax

DSPNOX:

Reverse Polish = 00650606

Common = DSPnox

SRCINVERT:

Reverse Polish = 00660046

Common = DSx

SDPSONOX:

Reverse Polish = 006718A8

Common = SDPSonox

DSPDSONOXXN:

Reverse Polish = 006858A6

Common = DSPDSonoxxn

PDSXXN:

Reverse Polish = 00690145

Common = PDSxxn

DPSAX:

Reverse Polish = 006A01E9

Common = DPSax

PSDPSOAXXN:

Reverse Polish = 006B178A

Common = PSDPSoaxxn

SDPAX:

Reverse Polish = 006C01E8

Common = SDPax

PDSPDOAXXN:

Reverse Polish = 006D1785

Common = PDSPDoaxxn

SDPSNOAX:

Reverse Polish = 006E1E28

Common = SDPSnoax

PDXNAN:

Reverse Polish = 006F0C65

Common = PDSxnan

PDSANA:

Reverse Polish = 00700CC5

Common = PDSana

SSDXPDXAXN:

Reverse Polish = 00711D5C

Common = SSDxPDxaxn

SDPSXOX:

Reverse Polish = 00720648

Common = SDPSxox

SDPNOAN:

Reverse Polish = 00730E28

Common = SDPnoan

DSPDXOX:

Reverse Polish = 00740646

Common = DSPDxox

DSPNOAN:

Reverse Polish = 00750E26

Common = DSPnoan

SDPSNAOX:

Reverse Polish = 00761B28

Common = SDPSnaox

DSAN:

Reverse Polish = 007700E6

Common = DSan

PDSAX:

Reverse Polish = 007801E5

Common = PDSax

DSPDSOAXXN:

Reverse Polish = 00791786

Common = DSPDSoaxxn

DPSDNOAX:

Reverse Polish = 007A1E29

Common = DPSDnoax

SDPXNAN:

Reverse Polish = 007B0C68

Common = SDPxnan

SPDSNOAX:

Reverse Polish = 007C1E24

Common = SPDSnoax

DPSXNAN:

Reverse Polish = 007D0C69

Common = DPSxnan

SPXDSXO:

Reverse Polish = 007E0955

Common = SPxDSxo

DPSAAN:

Reverse Polish = 007F03C9

Common = DPSaan

DPSAA:

Reverse Polish = 008003E9

Common = DPSaa

SPXDSXON:

Reverse Polish = 00810975

Common = SPxDSxon

DPSXNA:

Reverse Polish = 00820C49

Common = DPSxna

SPDSNOAXN:

Reverse Polish = 00831E04

Common = SPDSnoaxn

SDPXNA:

Reverse Polish = 00840C48

Common = SDPxna

PDSPNOAXN:

Reverse Polish = 00851E05

Common = PDSPnoaxn

DSPDSOAXX:

Reverse Polish = 008617A6

Common = DSPDSoaxx

PDSAXN:

Reverse Polish = 008701C5

Common = PDSaxn

SRCAND:

Reverse Polish = 008800C6

Common = DSa

SDPSNAOXN:

Reverse Polish = 00891B08

Common = SDPSnaoxn

DSPNOA:

Reverse Polish = 008A0E06

Common = DSPnoa

DSPDXOXN:

Reverse Polish = 008B0666

Common = DSPDxoxn

SDPNOA:

Reverse Polish = 008C0E08

Common = SDPnoa

SDPSXOXN:

Reverse Polish = 008D0668

Common = SDPSxoxn

SSDXPDXAX:

Reverse Polish = 008E1D7C

Common = SSDxPDxax

PDSANAN:

Reverse Polish = 008F0CE5

Common = PDSanan

PDSXNA:

Reverse Polish = 00900C45

Common = PDSxna

SDPSNOAXN:

Reverse Polish = 00911E08

Common = SDPSnoaxn

DPSDPOAXX:

Reverse Polish = 009217A9

Common = DPSDPoaxx

SPDAXN:

Reverse Polish = 009301C4

Common = SPDaxn

PSDPSOAXX:

Reverse Polish = 009417AA

Common = PSDPSoaxx

DPSAXN:

Reverse Polish = 009501C9

Common = DPSaxn

DPSXX:

Reverse Polish = 00960169

Common = DPSxx

PSDPSONOXX:

Reverse Polish = 0097588A

Common = PSDPSonoxx

SDPSONOXN:

Reverse Polish = 00981888

Common = SDPSonoxn

DSXN:

Reverse Polish = 00990066

Common = DSxn

DPSNAX:

Reverse Polish = 009A0709

Common = DPSnax

SDPSOAXN:

Reverse Polish = 009B07A8

Common = SDPSoaxn

SPDNAX:

Reverse Polish = 009C0704

Common = SPDnax

DSPDOAXN:

Reverse Polish = 009D07A6

Common = DSPDoaxn

DSPDSAOXX:

Reverse Polish = 009E16E6

Common = DSPDSaoxx

PDSXAN:

Reverse Polish = 009F0345

Common = PDSxan

DPA:

Reverse Polish = 00A000C9

Common = DPa

PDSPNAOXN:

Reverse Polish = 00A11B05

Common = PDSPnaoxn

DPSNOA:

Reverse Polish = 00A20E09

Common = DPSnoa

DPSDXOXN:

Reverse Polish = 00A30669

Common = DPSDxoxn

PDSPONOXN:

Reverse Polish = 00A41885

Common = PDSPonoxn

PDXN:

Reverse Polish = 00A50065

Common = PDxn

DSPNAX:

Reverse Polish = 00A60706

Common = DSPnax

PDSPOAXN:

Reverse Polish = 00A707A5

Common = PDSPoaxn

DPSOA:

Reverse Polish = 00A803A9

Common = DPSoa

DPSOXN:

Reverse Polish = 00A90189

Common = DPSoxn

D:

Reverse Polish = 00AA0029

Common = D

DPSONO:

Reverse Polish = 00AB0889

Common = DPSono

SPDSXAX:

Reverse Polish = 00AC0744

Common = SPDSxax

DPSDAOXN:

Reverse Polish = 00AD06E9

Common = DPSDaoxn

DSPNAO:

Reverse Polish = 00AE0B06

Common = DSPnao

DPNO:

Reverse Polish = 00AF0229

Common = DPno

PDSNOA:

Reverse Polish = 00B00E05

Common = PDSnoa

PDSPXOXN:

Reverse Polish = 00B10665

Common = PDSPxoxn

SSPXDSXOX:

Reverse Polish = 00B21974

Common = SSPxDSxox

SDPANAN:

Reverse Polish = 00B30CE8

Common = SDPanan

PSDNAX:

Reverse Polish = 00B4070A

Common = PSDnax

DPSDOAXN:

Reverse Polish = 00B507A9

Common = DPSDoaxn

DPSDPAOXX:

Reverse Polish = 00B616E9

Common = DPSDPaoxx

SDPXAN:

Reverse Polish = 00B70348

Common = SDPxan

PSDPXAX:

Reverse Polish = 00B8074A

Common = PSDPxax

DSPDAOXN:

Reverse Polish = 00B906E6

Common = DSPDaoxn

DPSNAO:

Reverse Polish = 00BA0B09

Common = DPSnao

MERGEPAINT:

Reverse Polish = 00BB0226

Common = DSno

SPDSANAX:

Reverse Polish = 00BC1CE4

Common = SPDSanax

SDXPDXAN:

Reverse Polish = 00BD0D7D

Common = SDxPDxan

DPSXO:

Reverse Polish = 00BE0269

Common = DPSxo

DPSANO:

Reverse Polish = 00BF08C9

Common = DPSano

MERGECOPY:

Reverse Polish = 00C000CA

Common = PSa

SPDSNAOXN:

Reverse Polish = 00C11B04

Common = SPDSnaoxn

SPDSONOXN:

Reverse Polish = 00C21884

Common = SPDSonoxn

PSXN:

Reverse Polish = 00C3006A

Common = PSxn

SPDNOA:

Reverse Polish = 00C40E04

Common = SPDnoa

SPDSXOXN:

Reverse Polish = 00C50664

Common = SPDSxoxn

SDPNAX:

Reverse Polish = 00C60708

Common = SDPnax

PSDPOAXN:

Reverse Polish = 00C707AA

Common = PSDPoaxn

SDPOA:

Reverse Polish = 00C803A8

Common = SDPoa

SPDOXN:

Reverse Polish = 00C90184

Common = SPDoxn

DPSDXAX:

Reverse Polish = 00CA0749

Common = DPSDxax

SPDSAOXN:

Reverse Polish = 00CB06E4

Common = SPDSaoxn

SRCCOPY:

Reverse Polish = 00CC0020

Common = S

SDPONO:

Reverse Polish = 00CD0888

Common = SDPono

SDPNAO:

Reverse Polish = 00CE0B08

Common = SDPnao

SPNO:

Reverse Polish = 00CF0224

Common = SPno

PSDNOA:

Reverse Polish =00D00E0A

Common = PSDnoa

PSDPXOXN:

Reverse Polish = 00D1066A

Common = PSDPxoxn

PDSNAX:

Reverse Polish = 00D20705

Common = PDSnax

SPDSOAXN:

Reverse Polish = 00D307A4

Common = SPDSoaxn

SSPXPDXAX:

Reverse Polish = 00D41D78

Common = SSPxPDxax

DPSANAN:

Reverse Polish = 00D50CE9

Common = DPSanan

PSDPSAOXX:

Reverse Polish = 00D616EA

Common = PSDPSaoxx

DPSXAN:

Reverse Polish = 00D70349

Common = DPSxan

PDSPXAX:

Reverse Polish = 00D80745

Common = PDSPxax

SDPSAOXN:

Reverse Polish = 00D906E8

Common = SDPSaoxn

DPSDANAX:

Reverse Polish = 00DA1CE9

Common = DPSDanax

SPXDSXAN:

Reverse Polish = 00DB0D75

Common = SPxDSxan

SPDNAO:

Reverse Polish = 00DC0B04

Common = SPDnao

SDNO:

Reverse Polish = 00DD0228

Common = SDno

SDPXO:

Reverse Polish = 00DE0268

Common = SDPxo

SDPANO:

Reverse Polish = 00DF08C8

Common = SDPano

PDSOA:

Reverse Polish = 00E003A5

Common = PDSoa

PDSOXN:

Reverse Polish = 00E10185

Common = PDSoxn

DSPDXAX:

Reverse Polish = 00E20746

Common = DSPDxax

PSDPAOXN:

Reverse Polish = 00E306EA

Common = PSDPaoxn

SDPSXAX:

Reverse Polish = 00E40748

Common = SDPSxax

PDSPAOXN:

Reverse Polish = 00E506E5

Common = PDSPaoxn

SDPSANAX:

Reverse Polish = 00E61CE8

Common = SDPSanax

SPXPDXAN:

Reverse Polish = 00E70D79

Common = SPxPDxan

SSPXDSXAX:

Reverse Polish = 00E81D74

Common = SSPxDSxax

DSPDSANAXXN:

Reverse Polish = 00E95CE6

Common = DSPDSanaxxn

DPSAO:

Reverse Polish = 00EA02E9

Common = DPSao

DPSXNO:

Reverse Polish = 00EB0849

Common = DPSxno

SDPAO:

Reverse Polish = 00EC02E8

Common = SDPao

SDPXNO:

Reverse Polish = 00ED0848

Common = SDPxno

SRCPAINT:

Reverse Polish = 00EE0086

Common = DSo

SDPNOO:

Reverse Polish = 00EF0A08

Common = SDPnoo

PATCOPY:

Reverse Polish = 00F00021

Common = P

PDSONO:

Reverse Polish = 00F10885

Common = PDSono

PDSNAO:

Reverse Polish = 00F20B05

Common = PDSnao

PSNO:

Reverse Polish = 00F3022A

Common = PSno

PSDNAO:

Reverse Polish = 00F40B0A

Common = PSDnao

PDNO:

Reverse Polish = 00F50225

Common = PDno

PDSXO:

Reverse Polish = 00F60265

Common = PDSxo

PDSANO:

Reverse Polish = 00F708C5

Common = PDSano

PDSAO:

Reverse Polish = 00F802E5

Common = PDSao

PDSXNO:

Reverse Polish = 00F90845

Common = PDSxno

DPO:

Reverse Polish = 00FA0089

Common = DPo

PATPAINT:

Reverse Polish = 00FB0A09

Common = DPSnoo

PSO:

Reverse Polish = 00FC008A

Common = PSo

PSDNOO:

Reverse Polish = 00FD0A0A

Common = PSDnoo

DPSOO:

Reverse Polish = 00FE02A9

Common = DPSoo

WHITENESS:

Reverse Polish = 00FF0062

Common = 1

Each ternary raster operation code represents a Boolean operation in which the values of the pixels in the source, the selected brush, and the destination are combined. Following are the three operands used in these operations:

Operand	Meaning
D	Destination bitmap

Operand	Meaning
P	Selected brush (also called pattern)
S	Source bitmap

Following are the Boolean operators used in these operations:

Operator	Meaning
a	Bitwise AND
n	Bitwise NOT (inverse)
o	Bitwise OR
x	Bitwise exclusive OR (XOR)

All Boolean operations are presented in reverse Polish notation. For example, the following operation replaces the values of the pixels in the destination bitmap with a combination of the pixel values of the source and brush: PSo.

The following operation combines the values of the pixels in the source and brush with the pixel values of the destination bitmap: DPSoo (there are alternative spellings of some functions, so although a particular spelling may not be listed in the enumeration, an equivalent form should be).

Each raster operation code is a 32-bit integer whose high-order word is a Boolean operation index and whose low-order word is the operation code. The 16-bit operation index is a zero-extended, 8-bit value that represents the result of the Boolean operation on predefined brush, source, and destination values. For example, the operation indexes for the PSo and DPSoo operations are shown in the following list:

P	S	D	PSo	DPSoo
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

The operation indexes are determined by reading the binary values in a column of the table from the bottom up. For example, in the PSo column, the binary value is 11111100, which is equivalent to 00FC (hexadecimal is implicit for these values), which is the operation index for PSo.

Using this method, DPSoo can be seen to have the operation index 00FE. Operation indexes define the locations of corresponding raster operation codes in the enumeration above. The PSo operation is in line 252 (0x00FC) of the enumeration; DPSoo is in line 254 (0x00FE).

The most commonly used raster operations have been given explicit enumeration names, which SHOULD be used; examples are **PATCOPY** and **WHITENESS**.

When the source and destination bitmaps are monochrome, a bit value of 0 represents a black pixel and a bit value of 1 represents a white pixel. When the source and the destination bitmaps are color, those colors are represented with red green blue (RGB) values.

2.1.33 TextAlignmentMode Enumeration

The Windows Metafile Format (WMF) **TextAlignmentMode Enumeration** specifies text alignment by using the values in the following enumeration, either individually or in combination. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

```
typedef  enum
{
  TA_NOUPDATECP = 0x0000,
  TA_LEFT = 0x0000,
  TA_TOP = 0x0000,
  TA_UPDATECP = 0x0001,
  TA_RIGHT = 0x0002,
  TA_CENTER = 0x0006,
  TA_BOTTOM = 0x0008,
  TA_BASELINE = 0x0018,
  TA_RTLREADING = 0x0100,
  VTA_TOP = TA_LEFT,
  VTA_RIGHT = TA_TOP,
  VTA_BOTTOM = TA_RIGHT,
  VTA_CENTER = TA_CENTER,
  VTA_LEFT = TA_BOTTOM,
  VTA_BASELINE = TA_BASELINE
} TextAlignmentMode;
```

TA_NOUPDATECP: The current position MUST NOT be updated after each text output call. The reference point MUST be passed to the text output function.

TA_LEFT: The reference point MUST be on the left edge of the bounding rectangle.

TA_TOP: The reference point MUST be on the top edge of the bounding rectangle.

TA_UPDATECP: The current position MUST be updated after each text output call. The current position MUST be used as the reference point.

TA_RIGHT: The reference point MUST be on the right edge of the bounding rectangle.

TA_CENTER: The reference point MUST be aligned horizontally with the center of the bounding rectangle.

TA_BOTTOM: The reference point MUST be on the bottom edge of the bounding rectangle.

TA_BASELINE: The reference point MUST be on the baseline of the text.

TA_RTLREADING: The text MUST be laid out in right-to-left reading order, instead of the default left-to-right order. This SHOULD be applied only when the font selected into the playback device context is either Hebrew or Arabic.

VTA_TOP: The reference point MUST be on the top edge of the bounding rectangle.

VTA_RIGHT: The reference point MUST be on the right edge of the bounding rectangle.

VTA_BOTTOM: The reference point MUST be on the bottom edge of the bounding rectangle.

VTA_CENTER: The reference point MUST be aligned vertically with the center of the bounding rectangle.

VTA_LEFT: The reference point MUST be on the left edge of the bounding rectangle.

VTA_BASELINE: The reference point MUST be on the baseline of the text.

When the current font has a vertical default baseline, as with Kanji, the VTA enumeration values MUST be used instead of TA values where there exists an equivalent.

Both TA and VTA values are named relative to their respective baselines. Thus, VTA_LEFT is the same as TA_BOTTOM, because the bottom edge of the bounding rectangle in normal text orientation becomes the left edge of the bounding rectangle with orientation relative to the vertical baseline.

TextAlignment mode specifies three different components of text alignment:

- The horizontal position of the reference point is determined by TA_RIGHT and TA_CENTER; if those bits are clear, the alignment MUST be TA_LEFT;
- The vertical position of the reference point is determined by TA_BOTTOM and TA_BASELINE; if those bits are clear, the alignment MUST be TA_TOP; and
- Whether to update the current position after text output is determined by TA_UPDATECP; if that bit is clear, the position MUST NOT be updated.

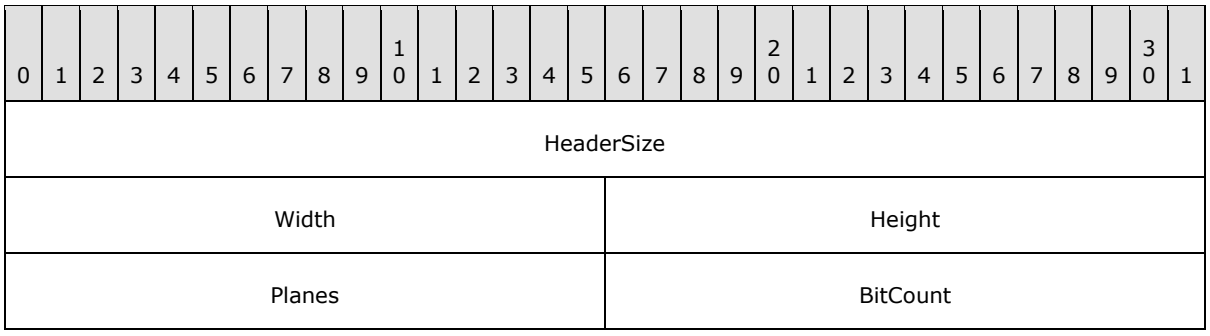
This is the reason for defining three different zero values in the enumeration; they represent the default states of the three components of text alignment.

2.2 WMF Objects

2.2.1 Fixed-Length Objects

2.2.1.1 BitmapCoreHeader Object

The Windows Metafile Format (WMF) BitmapCoreHeader Object contains information about the dimensions and color format of a DeviceIndependentBitmap (DIB) Object.



HeaderSize (4 bytes): A 32-bit unsigned integer that defines the size of this object, in bytes.

Width (2 bytes): A 16-bit unsigned integer that defines the width of the bitmap, in pixels.

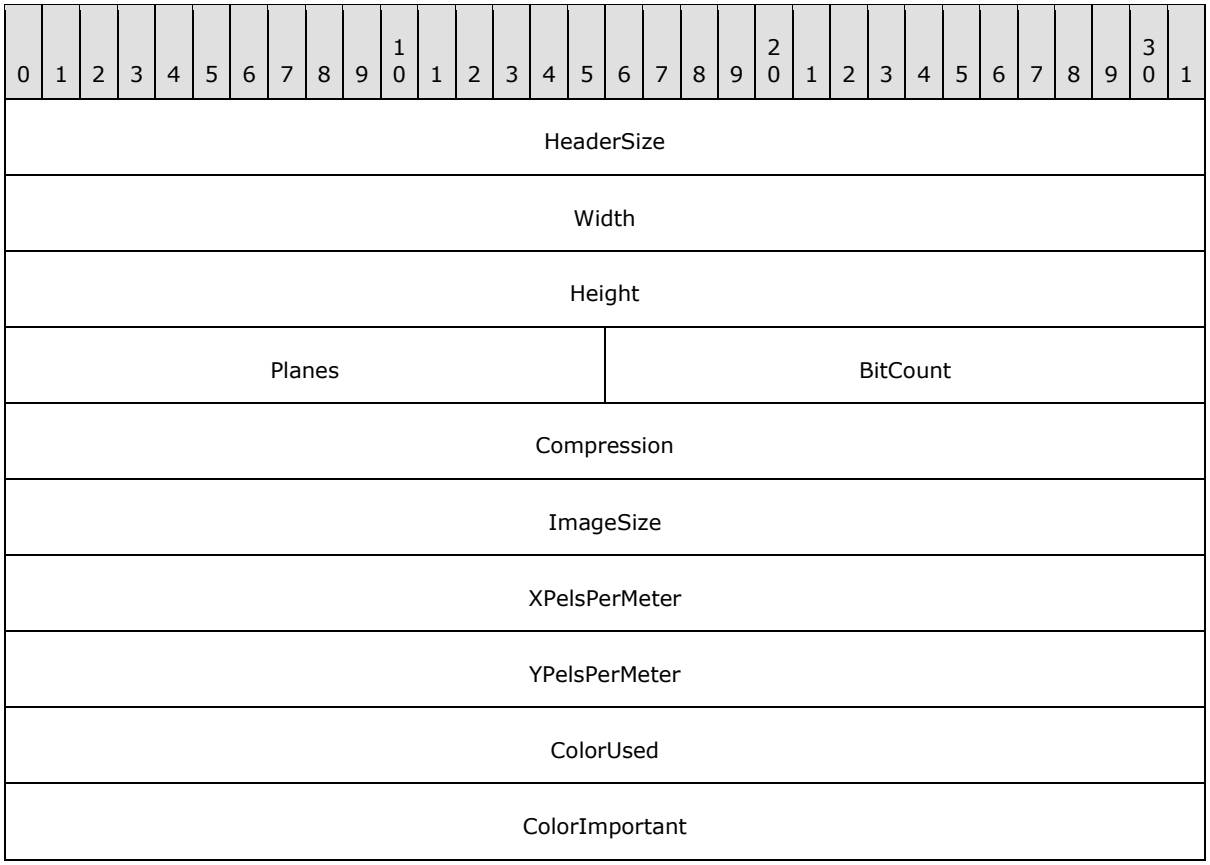
Height (2 bytes): A 16-bit unsigned integer that defines the height of the bitmap, in pixels.

Planes (2 bytes): A 16-bit unsigned integer that defines the number of planes for the target device. This value MUST be 0x0001.

BitCount (2 bytes): A 16-bit unsigned integer that defines the number of bits-per-pixel, the number of bits that define each pixel, and the maximum number of colors in the bitmap. The value MUST be one of the values in the **BitCount Enumeration** table.

2.2.1.2 BitmapInfoHeader Object

The Windows Metafile Format (WMF) BitmapInfoHeader Object contains information about the dimensions and color format of a DeviceIndependentBitmap (DIB) Object.



HeaderSize (4 bytes): A 32-bit unsigned integer that defines the size of this object, in bytes.

Width (4 bytes): A 32-bit signed integer that defines the width of the bitmap, in pixels.

If the **Compression** field is BI_JPEG or BI_PNG, this field specifies the width of the decompressed JPEG or PNG image file, respectively.

Height (4 bytes): A 32-bit signed integer that defines the height of the bitmap, in pixels. If **Height** is positive, the bitmap is a bottom-up DIB and its origin is the lower-left corner. If **Height** is negative, the bitmap is a top-down DIB and its origin is the upper-left corner.

If **Height** is negative, indicating a top-down DIB, **Compression** MUST be either BI_RGB or BI_BITFIELDS. Top-down DIBs cannot be compressed.

If **Compression** is BI_JPEG or BI_PNG, the **Height** field specifies the height of the decompressed JPEG or PNG image file, respectively.

Planes (2 bytes): A 16-bit unsigned integer that defines the number of planes for the target device. This value MUST be 0x0001.

BitCount (2 bytes): A 16-bit unsigned integer that defines the number bits that define each pixel and the maximum number of colors in the bitmap. This MUST be one of the values in the **BitCount Enumeration (section 2.1.3)**.

Compression (4 bytes): A 32-bit unsigned integer that defines the type of compression for a compressed bottom-up bitmap (top-down DIBs cannot be compressed). This field can be one of the values in the **Compression Enumeration (section 2.1.8)**.

ImageSize (4 bytes): A 32-bit unsigned integer that defines the size, in bytes, of the image. This may be set to 0 for BI_RGB, BI_BITFIELDS, and BI_CMYK bitmaps.

If **Compression** is BI_JPEG or BI_PNG, **ImageSize** indicates the size of the JPEG or PNG image buffer, respectively.

XPelsPerMeter (4 bytes): A 32-bit signed integer that defines the horizontal resolution, in pixels-per-meter, of the target device for the bitmap.

YPelsPerMeter (4 bytes): A 32-bit signed integer that defines the vertical resolution, in pixels-per-meter, of the target device for the bitmap.

ColorUsed (4 bytes): A 32-bit unsigned integer that defines the number of color indexes in the color table that are actually used by the bitmap. If this value is 0, the bitmap uses the maximum number of colors corresponding to the value of the **BitCount** field for the compression mode specified by **Compression**.

If **ColorUsed** is nonzero and the **BitCount** field is less than 16, the **ColorUsed** field specifies the actual number of colors that the graphics engine or device driver accesses.

If **BitCount** is 16 or greater, the **ColorUsed** field specifies the size of the color table used to optimize performance of the system palettes.

If **BitCount** equals 16 or 32, the optimal color palette SHOULD start immediately following the three 16-bit unsigned integer masks.

When the bitmap array immediately follows the BitmapInfoHeader Object, it is called a **"packed" bitmap**. In Packed bitmaps, the **ColorUsed** field MUST be either 0 or the actual size of the color table.

ColorImportant (4 bytes): A 32-bit unsigned integer that defines the number of color indexes that are required for displaying the bitmap. If this value is 0, all colors are required.

2.2.1.3 BitmapV4Header Object

The Windows Metafile Format BitmapV4Header Object contains information about the dimensions and color format of a DIB. It is an extended version of the BitmapInfoHeader Object (section 2.2.1.2).<18>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
BitmapInfoHeader																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(BitmapInfoHeader cont'd for 2 rows)																															
RedMask																															
GreenMask																															
BlueMask																															
AlphaMask																															
ColorSpaceType																															
Endpoints																															
...																															
...																															
...																															
...																															

...
...
...
...
GammaRed
GammaGreen
GammaBlue

BitmapInfoHeader (40 bytes): A 320-bit BitmapInfoHeader Object that defines the header information of this object.

RedMask (4 bytes): A 32-bit unsigned integer that defines the color mask that specifies the red component of each pixel, valid only if **Compression** is set to BI_BITFIELDS.

GreenMask (4 bytes): A 32-bit unsigned integer that defines the color mask that specifies the green component of each pixel, valid only if **Compression** is set to BI_BITFIELDS.

BlueMask (4 bytes): A 32-bit unsigned integer that defines the color mask that specifies the blue component of each pixel, valid only if **Compression** is set to BI_BITFIELDS.

AlphaMask (4 bytes): A 32-bit unsigned integer that defines the color mask that specifies the alpha component of each pixel.

ColorSpaceType (4 bytes): A 32-bit unsigned integer that defines the color space of the DIB Object. This value MUST be LCS_CALIBRATED_RGB from the **LogicalColorSpace Enumeration** table, which indicates that endpoints and **gamma** values are given in the appropriate fields.

See the LogColorSpace Objects, sections 2.2.2.5 and 2.2.2.6, for information that defines a logical color space.

Endpoints (36 bytes): A 288-bit CIEXYZTriple Object that defines the **CIE** chromaticity x, y, and z coordinates of the three colors that correspond to the red, green, and blue endpoints for the logical color space associated with the bitmap. This member is ignored unless the **ColorSpaceType** field specifies LCS_CALIBRATED_RGB.

GammaRed (4 bytes): A 32-bit fixed point that defines the toned response curve for red. This member is ignored unless color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

GammaGreen (4 bytes): A 32-bit fixed point that defines the toned response curve for green. This member is ignored unless color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

GammaBlue (4 bytes): A 32-bit fixed point that defines the toned response curve for blue. This member is ignored unless color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

The gamma member format is an unsigned "8.8" fixed **point** integer that is then left-shifted by 8 bits. "8.8" means "8 integer bits followed by 8 fraction bits": nnnnnnnnffffffff. Taking the shift into account, the required format of the 32-bit DWORD is: 00000000nnnnnnnnffffffff00000000.

2.2.1.4 BitmapV5Header Object

The Windows Metafile Format BitmapV5Header Object contains information about the dimensions and color format of a DIB. It is an extended version of the BitmapV4Header Object (section 2.2.1.3).<19>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
BitmapV4Header																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(BitmapV4Header cont'd for 19 rows)																															
Intent																															
ProfileData																															
ProfileSize																															
Reserved																															

BitmapV4Header (108 bytes): A BitmapV4Header Object, specified in section 2.2.1.3 , which defines header information of this object.

When it is part if a BitmapV5Header Object, the **ColorSpaceType** field of a BitmapV4Header Object MAY be a logical color space value in the **LogicalColorSpaceV5 Enumeration** table.

Intent (4 bytes): A 32-bit unsigned integer that defines the rendering intent for bitmap. This MAY be defined in the **LogicalColorSpace Enumeration** table.

ProfileData (4 bytes): A 32-bit unsigned integer that defines the offset, in bytes, from the beginning of the BitmapV5Header Object to the start of the color profile data.

If the color profile is embedded in the DIB, **ProfileData** is the actual color profile; if the color profile is linked, the **ProfileData** is the null-terminated file name of the color profile. This MUST NOT be a Unicode string, but MUST be composed exclusively of characters from the Windows character set (code page 1252).

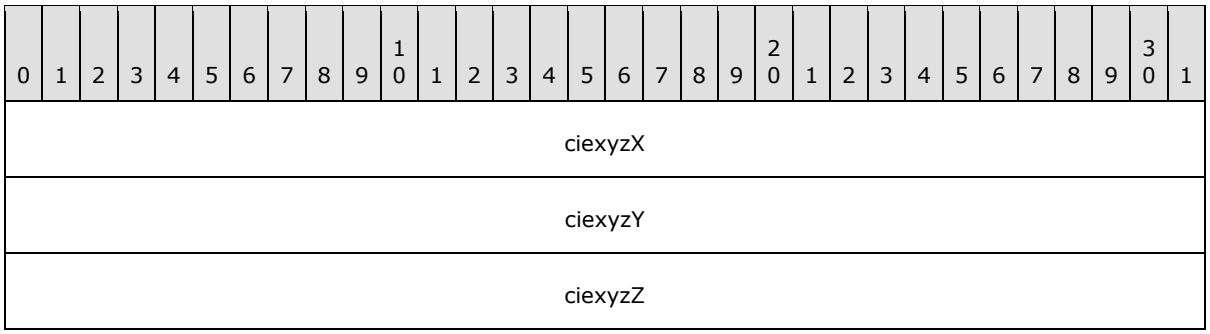
These color profile members are ignored unless the **ColorSpaceType** field in BitmapV4Header Object specifies LCS_PROFILE_LINKED or LCS_PROFILE_EMBEDDED.

ProfileSize (4 bytes): A 32-bit unsigned integer that defines the size, in bytes, of embedded color profile data.

Reserved (4 bytes): A 32-bit unsigned integer that is undefined and SHOULD be ignored.

2.2.1.5 CIEXYZ Object

The Windows Metafile Format CIEXYZ Object defines information about the **CIEXYZ** chromaticity object.



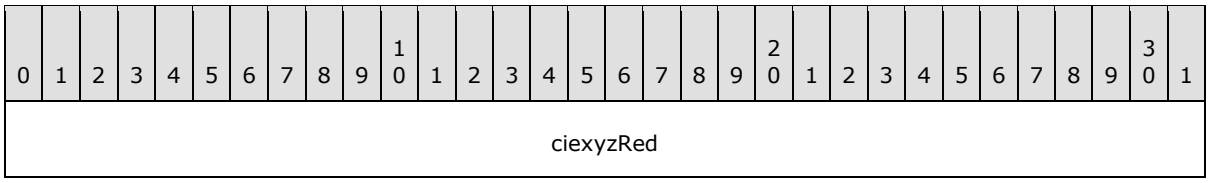
ciexyzX (4 bytes): A 32-bit 2.30 fixed point type that defines the x chromaticity value.

ciexyzY (4 bytes): A 32-bit 2.30 fixed point type that defines the y chromaticity value.

ciexyzZ (4 bytes): A 32-bit 2.30 fixed point type that defines the z chromaticity value.

2.2.1.6 CIEXYZTriple Object

The Windows Metafile Format CIEXYZTriple Object defines information about the CIEXYZTriple color object.



...
...
ciexyzGreen
...
...
ciexyzBlue
...
...

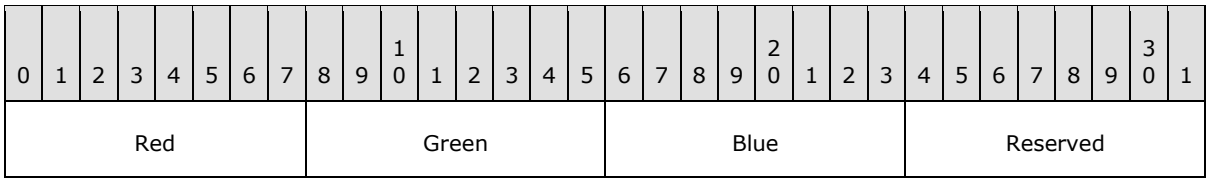
ciexyzRed (12 bytes): A 96-bit CIEXYZ Object that defines the red chromaticity values.

ciexyzGreen (12 bytes): A 96-bit CIEXYZ Object that defines the green chromaticity values.

ciexyzBlue (12 bytes): A 96-bit CIEXYZ Object that defines the blue chromaticity values.

2.2.1.7 ColorRef Object

The Windows Metafile Format ColorRef Object defines the RGB color.



Red (1 byte): An 8-bit unsigned integer that defines the relative intensity of red.

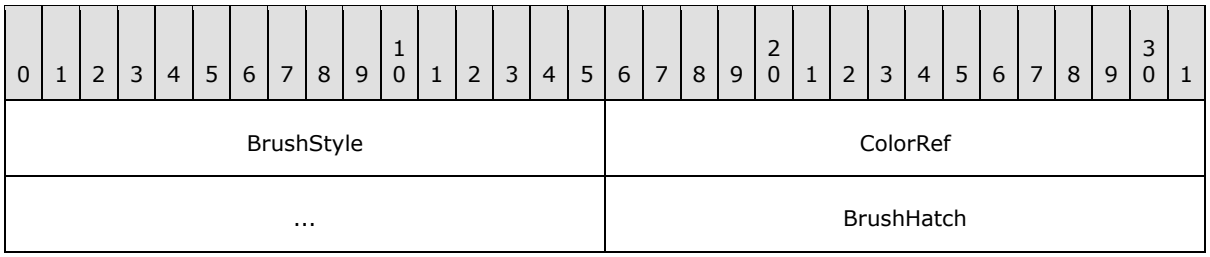
Green (1 byte): An 8-bit unsigned integer that defines the relative intensity of green.

Blue (1 byte): An 8-bit unsigned integer that defines the relative intensity of blue.

Reserved (1 byte): An 8-bit unsigned integer that MUST be 0x00.

2.2.1.8 LogBrush Object

The Windows Metafile Format (WMF) LogBrush Object defines the style, color, and pattern of a brush. This object is used only in the META_CREATEBRUSHINDIRECT (section 2.3.4.1) record to create a Brush Object (section 2.2.2.2).



BrushStyle (2 bytes): A 16-bit unsigned integer that defines the brush style. The value MUST be an enumeration from the **BrushStyle Enumeration** table. For the meanings of the different values, see the table below. The BS_NULL style SHOULD be used to specify a brush that has no effect. <20>

ColorRef (4 bytes): A 32-bit ColorRef Object, specified in section 2.2.1.7 , that specifies a color. Its interpretation depends on the value of **BrushStyle**, as explained below.

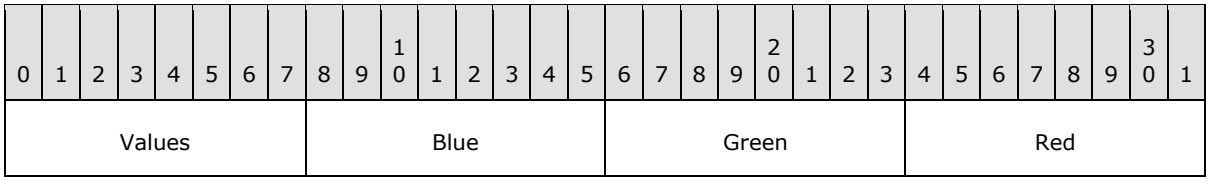
BrushHatch (2 bytes): A 16-bit field that contains the brush hatch type. Its interpretation depends on the value of **BrushStyle**, as explained below.

The following table shows the relationship between the **BrushStyle**, **ColorRef** and **BrushHatch** fields in a LogBrush Object. Only supported brush styles are listed.

BrushStyle	ColorRef	BrushHatch
BS_SOLID	SHOULD be a ColorRef Object, which determines the color of the brush.	Not used, and SHOULD be ignored.
BS_NULL	Not used, and SHOULD be ignored.	Not used, and SHOULD be ignored.
BS_PATTERN	Not used, and SHOULD be ignored.	Not used. A default object, such as a solid-color black Brush Object, MAY be created.<21>
BS_DIBPATTERN	Not used, and SHOULD be ignored.	Not used. A default object, such as a solid-color black Brush Object, MAY be created.
BS_DIBPATTERNPT	Not used, and SHOULD be ignored.	Not used. A default object, such as a solid-color black Brush Object, MAY be created.
BS_HATCHED	SHOULD be a ColorRef Object, which determines the foreground color of the hatch pattern.	A value from the **HatchStyle Enumeration** that specifies the orientation of lines used to create the hatch.

2.2.1.9 PaletteEntry Object

The Windows Metafile Format PaletteEntry Object defines the color and usage of an entry in a palette.



Values (1 byte): An 8-bit unsigned integer that defines how the palette entry is to be used. The **Values** field MUST be 0x00 or one of the values in the **PaletteEntryFlag Enumeration** table.

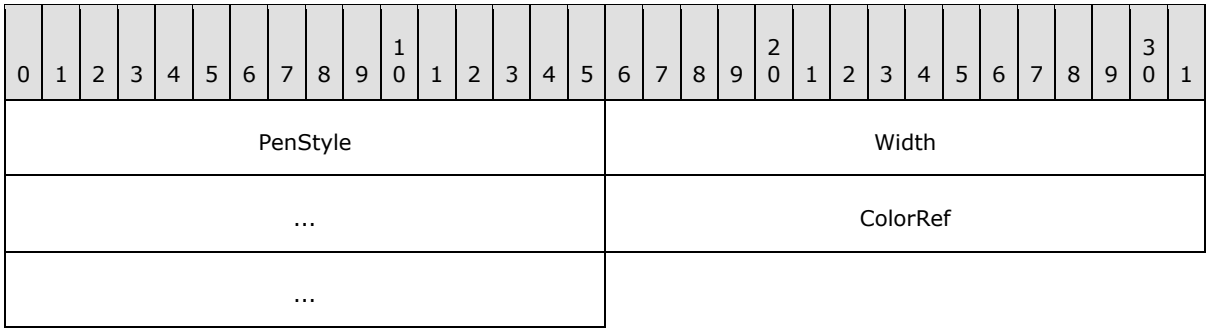
Blue (1 byte): An 8-bit unsigned integer that defines the blue intensity value for the palette entry.

Green (1 byte): An 8-bit unsigned integer that defines the green intensity value for the palette entry.

Red (1 byte): An 8-bit unsigned integer that defines the red intensity value for the palette entry.

2.2.1.10 Pen Object

The Windows Metafile Format (WMF) Pen Object specifies the style, width, and color of a pen.



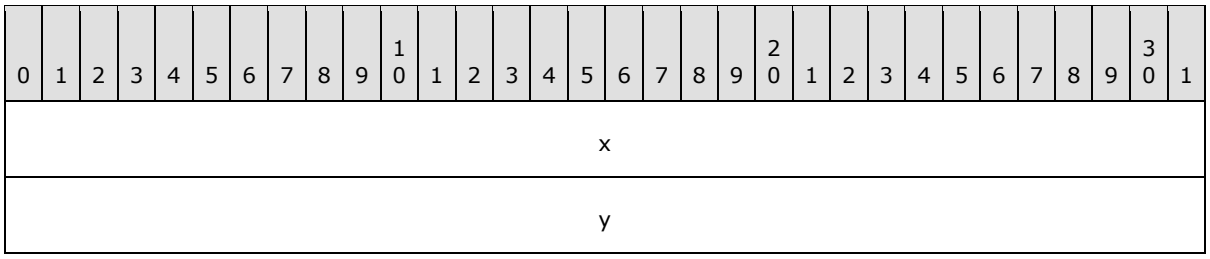
PenStyle (2 bytes): A 16-bit unsigned integer that specifies the pen style. The value MUST be defined from the **PenStyle Enumeration** table.

Width (4 bytes): A 32-bit PointS Object that specifies a point for the object dimensions. The x coordinate is the pen width. The y-coordinate is ignored.

ColorRef (4 bytes): A 32-bit ColorRef Object that specifies the pen color value.

2.2.1.11 PointL Object

The Windows Metafile Format (WMF) PointL Object defines the coordinates of a point.

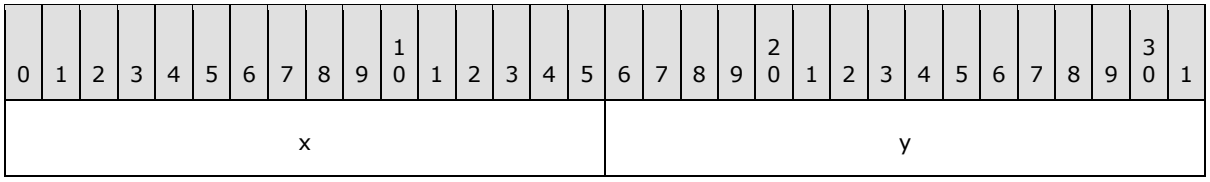


x (4 bytes): A 32-bit signed integer that defines the horizontal (x) coordinate of the point.

y (4 bytes): A 32-bit signed integer that defines the vertical (y) coordinate of the point.

2.2.1.12 PointS Object

The Windows Metafile Format PointS Object defines the x- and y-coordinates of a point.

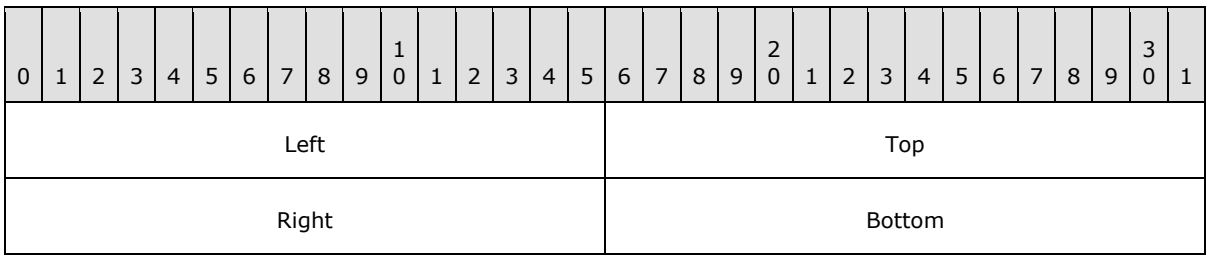


x (2 bytes): A 16-bit signed integer that defines the horizontal (x) coordinate of the point.

y (2 bytes): A 16-bit signed integer that defines the vertical (y) coordinate of the point.

2.2.1.13 Rect Object

The Windows Metafile Format (WMF) Rect Object defines a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.



Left (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical coordinates, of the upper-left corner of the rectangle

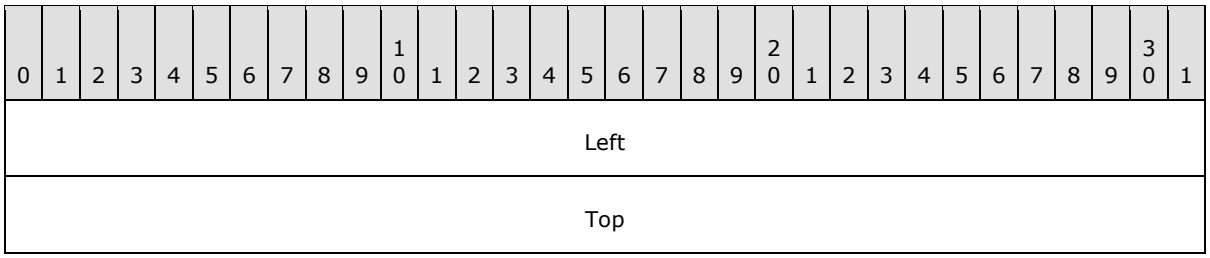
Top (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical coordinates, of the upper-left corner of the rectangle.

Right (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical coordinates, of the lower-right corner of the rectangle.

Bottom (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical coordinates, of the lower-right corner of the rectangle.

2.2.1.14 RectL Object

The Windows Metafile Format (WMF) RectL Object defines a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.



Right
Bottom

Left (4 bytes): A 32-bit signed integer that defines the x coordinate, in logical coordinates, of the upper-left corner of the rectangle.

Top (4 bytes): A 32-bit signed integer that defines the y coordinate, in logical coordinates, of the upper-left corner of the rectangle.

Right (4 bytes): A 32-bit signed integer that defines the x coordinate, in logical coordinates, of the lower-right corner of the rectangle.

Bottom (4 bytes): A 32-bit signed integer that defines y coordinate, in logical coordinates, of the lower-right corner of the rectangle.

When the RectL Object is contained in the Rect Object, the rectangle is filled up to— but not including—the right column and bottom row of pixels.

2.2.1.15 RGBQuad Object

The Windows Metafile Format RGBQuad Object defines the pixel color values in an uncompressed DIB.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Blue								Green								Red								Reserved							

Blue (1 byte): An 8-bit unsigned integer that defines the relative intensity of blue.

Green (1 byte): An 8-bit unsigned integer that defines the relative intensity of green.

Red (1 byte): An 8-bit unsigned integer that defines the relative intensity of red.

Reserved (1 byte): An 8-bit unsigned integer that MUST be 0x00.

2.2.1.16 SizeL Object

The Windows Metafile Format SizeL Object defines the x- and y-extents of a rectangle.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cx																															
cy																															

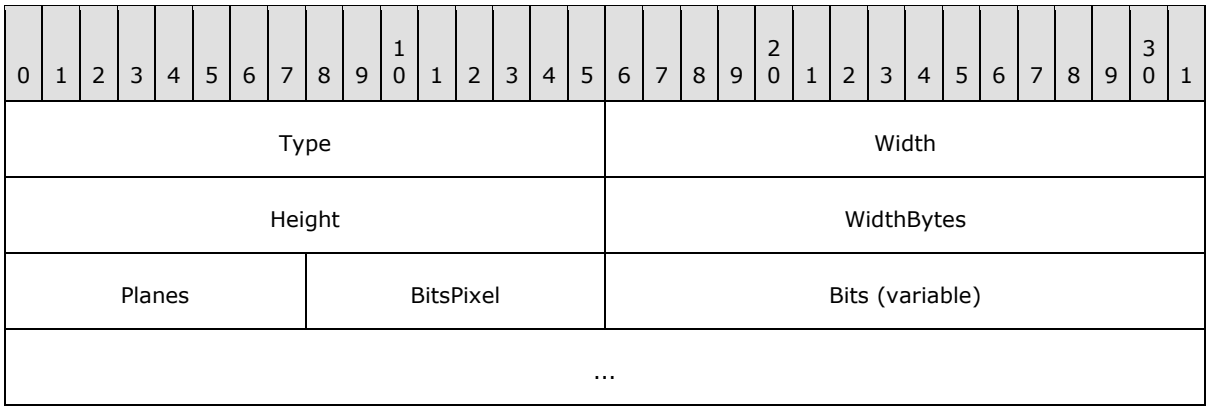
cx (4 bytes): A 32-bit unsigned integer that defines the x-coordinate of the point.

cy (4 bytes): A 32-bit unsigned integer that defines the y-coordinate of the point.

2.2.2 Variable-Length Objects

2.2.2.1 Bitmap16 Object

The Windows Metafile Format Bitmap16 Object specifies information about the dimensions and color format of a bitmap.



Type (2 bytes): A 16-bit signed integer that defines the bitmap type.

Width (2 bytes): A 16-bit signed integer that defines the width of the bitmap in pixels.

Height (2 bytes): A 16-bit signed integer that defines the height of the bitmap in scan lines.

WidthBytes (2 bytes): A 16-bit signed integer that defines the number of bytes per scan line.

Planes (1 byte): An 8-bit unsigned integer that defines the number of color planes in the bitmap. The value of this field MUST be 0x01.

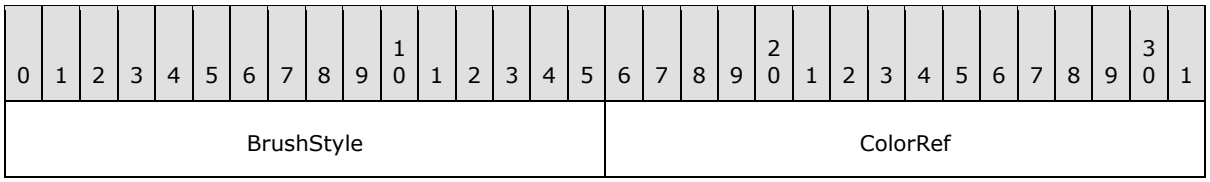
BitsPixel (1 byte): An 8-bit unsigned integer that defines the number of adjacent color bits on each plane.

Bits (variable): A variable length array of bytes that defines the bitmap pixel data. The length of this field in bytes can be computed as follows:

```
(((Width * BitsPixel + 15) >> 4) << 1) * Height
```

2.2.2.2 Brush Object

The Windows Metafile Format Brush Object defines the style, color, and pattern of a brush. Brush Objects are created by the META_CREATEBRUSHINDIRECT, META_CREATEPATTERNBRUSH and META_DIBCREATEPATTERNBRUSH records.



...	BrushHatch (variable)
...	

BrushStyle (2 bytes): A 16-bit unsigned integer that defines the brush style. The value MUST be an enumeration from the **BrushStyle Enumeration** table. For the meanings of the different values, see the table below.

ColorRef (4 bytes): A 32-bit field that specifies how to interpret color values in the object defined in the **BrushHatch** field. Its interpretation depends on the value of **BrushStyle**, as explained below.

BrushHatch (variable): A variable-size field that contains the brush hatch or pattern data. The content depends on the value of **BrushStyle**, as explained below.

The **BrushStyle** field determines how the **ColorRef** and **BrushHatch** fields SHOULD be interpreted, as specified in the following table.

The following table shows the relationship between the **BrushStyle**, **ColorRef** and **BrushHatch** fields in a Brush Object.

BrushStyle	ColorRef	BrushHatch
BS_SOLID	SHOULD be a ColorRef Object, specified in section 2.2.1.7.	Not used, and SHOULD be ignored.
BS_NULL	SHOULD be ignored.	Not used, and SHOULD be ignored.
BS_PATTERN	SHOULD be ignored.	SHOULD be a Bitmap16 Object, specified in section 2.2.2.1, which defines the brush pattern.
BS_DIBPATTERNPT	SHOULD be a 32-bit **ColorUsage Enumeration** value, specified in section 2.1.7; the low-order word specifies the meaning of color values in the DIB.	SHOULD be a DIB Object, specified in section 2.2.2.3, which defines the brush pattern.
BS_HATCHED	SHOULD be a ColorRef Object, specified in section 2.2.1.7.	SHOULD be a 16-bit value from the **HatchStyle Enumeration** table, specified in section 2.1.14, which defines the brush pattern.

2.2.2.3 DeviceIndependentBitmap Object

The Windows Metafile Format (WMF) DeviceIndependentBitmap (DIB) Object defines an image in a DIB. It includes header information about the dimensions and format of the image, an optional color table, and the image pixel data.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
DIBHeaderInfo (variable)																															

...
Colors (variable)
...
aData (variable)
...

DIBHeaderInfo (variable): Either a BitmapCoreHeader Object (section 2.2.1.1) or a BitmapInfoHeader Object (section 2.2.1.2) that defines the image header information.

In the first 32 bits of the **DIBHeaderInfo** field, if the **HeaderSize** value is 0x0000000C, then this is a BitmapCoreHeader Object. Otherwise, it is a BitmapInfoHeader Object.

Colors (variable): An optional array of either RGBQuad Objects (section 2.2.1.15) or 16-bit unsigned integers that define a color table.

If a DIB Object is defined within a WMF record or object that specifies a **ColorUsage Enumeration (section 2.1.7)** value, the size and contents of this array can be determined from that value and from the information in the **DIBHeaderInfo**. See **BitCount Enumeration (section 2.1.3)** for details.

If there is no **ColorUsage** value defined in the DIB container, DIB_RGB_COLORS is assumed, which means the color table contains an array of RGBQuad Objects.

aData (variable): An array of bytes that define the actual bitmap bits. The size and format of the bitmap data is determined by the **DIBHeaderInfo** field. When **DIBHeaderInfo** is a BitmapCoreHeader Object, the size in bytes of **aData** MUST be calculated from values in the BitmapCoreHeader Object, as follows:

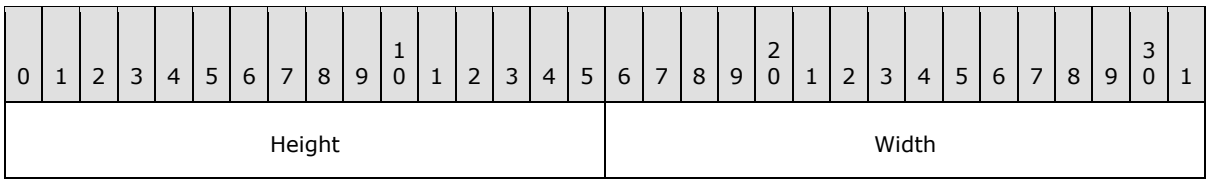
```
(((Width * Planes * BitCount + 31) & ~31) / 8) * abs(Height)
```

This formula SHOULD also be used to calculate the size of **aData** when **DIBHeaderInfo** is a BitmapInfoHeader Object, using values from that object, but only if its **Compression** value is BI_RGB, BI_BITFIELDS, or BI_CMYK.

Otherwise, the size of **aData** MUST be the BitmapInfoHeader Object value **ImageSize**.

2.2.2.4 Font Object

The Windows Metafile Format (WMF) Font Object specifies the attributes of a logical font.



Escapement		Orientation	
Weight		Italic	Underline
StrikeOut	CharSet	OutPrecision	ClipPrecision
Quality	PitchAndFamily	Facename (variable)	
...			

Height (2 bytes): A 16-bit signed integer that specifies the height, in logical units, of the font's character cell. The character height is computed as the character cell height minus the internal leading. The font mapper SHOULD interpret the height as follows:

Value	Meaning
0 <*value*	The font mapper SHOULD transform this value into device units and match it against the cell height of available fonts.
0	A default height value MUST be used when creating a physical font.
value < 0	The font mapper SHOULD transform this value into device units and match its absolute value against the character height of available fonts.

For all height comparisons, the font mapper SHOULD find the largest physical font that does not exceed the requested size. <22>

Width (2 bytes): A 16-bit signed integer that defines the average width, in logical units, of characters in the font. If **Width** is 0x0000, the aspect ratio of the device SHOULD be matched against the digitization aspect ratio of the available fonts to find the closest match, determined by the absolute value of the difference.

Escapement (2 bytes): A 16-bit signed integer that defines the angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text.

Orientation (2 bytes): A 16-bit signed integer that defines the angle, in tenths of degrees, between each character's base line and the x-axis of the device.

Weight (2 bytes): A 16-bit signed integer that defines the weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is 0x0000, a default weight SHOULD be used.

Italic (1 byte): A 8-bit unsigned integer that specifies the italic attribute of the font.

Value	Meaning
FALSE 0x00	This is not an italic font.
TRUE 0x01	This is an italic font.

Underline (1 byte): A 8-bit unsigned integer that specifies the underline attribute of the font.

Value	Meaning
FALSE 0x00	This is not an underline font.
TRUE 0x01	This is an underline font.

StrikeOut (1 byte): A 8-bit unsigned integer that specifies the strikeout attribute of the font.

Value	Meaning
FALSE 0x00	This is not a strikeout font.
TRUE 0x01	This is a strikeout font.

CharSet (1 byte): A 8-bit unsigned integer that defines the character set. It SHOULD be set to a value in the **CharacterSet Enumeration (section 2.1.5)**.

The DEFAULT_CHARSET value MAY be used to allow the name and size of a font to fully describe the logical font. If the specified font name does not exist, a font in another character set MAY be substituted. The DEFAULT_CHARSET value is set to a value based on the current system locale. For example, when the system locale is United States, it is set to ANSI_CHARSET.

If a typeface name in the **FaceName** field is specified, the **CharSet** value MUST match the character set of that typeface.

OutPrecision (1 byte): An 8-bit unsigned integer that defines the output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It MUST be one of the values from the **OutPrecision Enumeration (section 2.1.23)**.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper selects a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named "Symbol" in raster and TrueType forms, specifying OUT_TT_PRECIS forces the font mapper to select the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to select a TrueType font, even if it substitutes a TrueType font of another name.

ClipPrecision (1 byte): An 8-bit unsigned integer that defines the clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It MUST be a combination of one or more of the bit settings in the **ClipPrecision Enumeration (section 2.1.6)**.

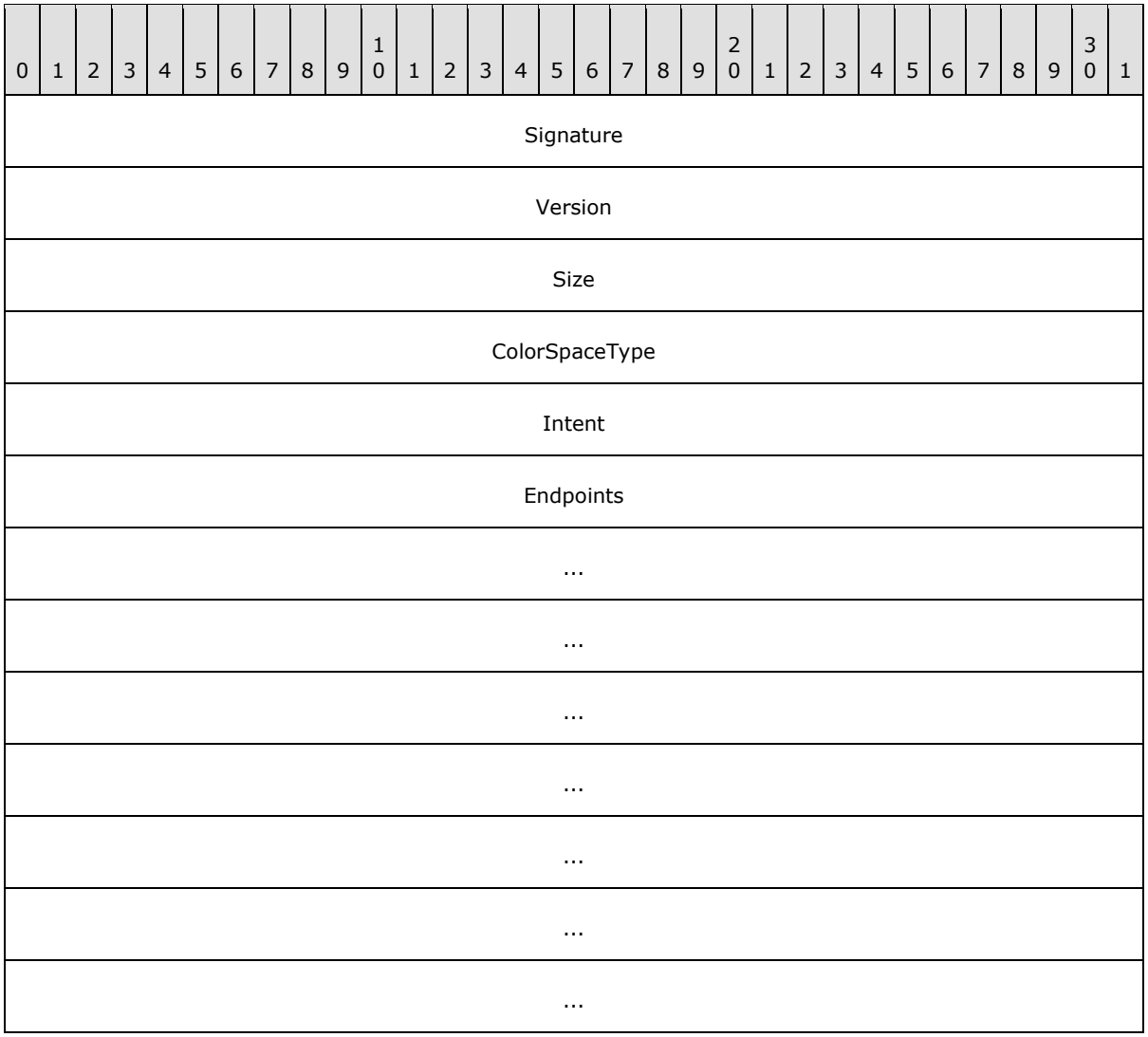
Quality (1 byte): An 8-bit unsigned integer that defines the output quality. The output quality defines how carefully to attempt to match the logical font attributes to those of an actual physical font. It MUST be one of the values in the **FontQuality Enumeration (section 2.1.12)**.

PitchAndFamily (1 byte): An 8-bit unsigned integer that defines the pitch and family of the font. The two low-order bits define the pitch of the font and MUST be one of the values in the **PitchFont Enumeration (section 2.1.26)**. Bits 4 through 7 define the font family and MUST be one of the values in the **FamilyFont Enumeration (section 2.1.10)**. Font families specify the look of a font in a general way. They are intended for specifying fonts when the exact typeface wanted is not available.

Facename (variable): A null-terminated string of 8-bit **ANSI character set** characters that specifies the typeface name of the font. The length of this string MUST NOT exceed 32 8-bit characters, including the terminating null.

2.2.2.5 LogColorSpace Object

The Windows Metafile Format (WMF) LogColorSpace Object defines information about the logical color space object with an ANSI character file name.



...
GammaRed
GammaGreen
GammaBlue
Filename (variable)
...

Signature (4 bytes): A 32-bit unsigned integer that specifies the signature of color space objects; it MUST be set to the value 0x50534F43, which is the ASCII encoding of the string "PSOC".

Version (4 bytes): A 32-bit unsigned integer that defines the version number; it MUST be 0x00000400.

Size (4 bytes): A 32-bit unsigned integer that defines the size of this object, in bytes.

ColorSpaceType (4 bytes): A 32-bit signed integer that defines the color space type. It MAY be defined in the **LogicalColorSpace Enumeration**.

These values are translated by using the endpoints specified by the **Endpoints** field before being passed to the device.

Intent (4 bytes): A 32-bit signed integer that defines the gamut mapping intent. It MAY be defined in the **GamutMappingIntent Enumeration**.

Endpoints (36 bytes): A 288-bit CIEXYZTriple Object that defines the Commission Internationale de l'Eclairage International (CIE) chromaticity x, y, and z coordinates of the three colors that correspond to the red green blue (RGB) endpoints for the logical color space associated with the bitmap. This field is ignored unless the **ColorSpaceType** field specifies LCS_CALIBRATED_RGB.

GammaRed (4 bytes): A 32-bit fixed-point value that defines the toned response curve for red. This field is ignored unless the color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

GammaGreen (4 bytes): A 32-bit fixed point that defines the toned response curve for green. This field is ignored unless the color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

GammaBlue (4 bytes): A 32-bit fixed-point value that defines the toned response curve for blue. This field is ignored unless the color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

The gamma field format is an unsigned "8.8" fixed-point integer that is then left-shifted by 8 bits. "8.8" means "8 integer bits followed by 8 fraction bits": nnnnnnnnffffffff. Taking the shift into account, the required format of the 32-bit **DWORD** is: 00000000nnnnnnnnffffffff00000000.

Filename (variable): A variable-length, ANSI character array that names a color profile file. This field MAY be used to set the color space to be exactly as specified by the color profile. This is useful for devices that input color values for a specific printer, or when using an installable image color matcher. If a color profile is specified, all other fields of this structure SHOULD be set to reasonable values, even if the values are not completely accurate.

If the **ColorSpaceType** field is set to LCS_sRGB or LCS_WINDOWS_COLOR_SPACE, the other fields of this structure are ignored, and **Windows Color System (WCS)** uses the sRGB color space. The **Endpoints**, **GammaRed**, **GammaGreen**, and **GammaBlue** fields are used to describe the logical color space. The **Endpoints** field is a CIEXYZTriple Object that contains the x, y, and z values of the color space's RGB endpoint.

Whenever the **Filename** field contains a file name and the **ColorSpaceType** field is set to LCS_CALIBRATED_RGB, processing SHOULD ignore the other fields of this structure. It uses the color space in the file as the color space to which this LogColorSpace Object refers.

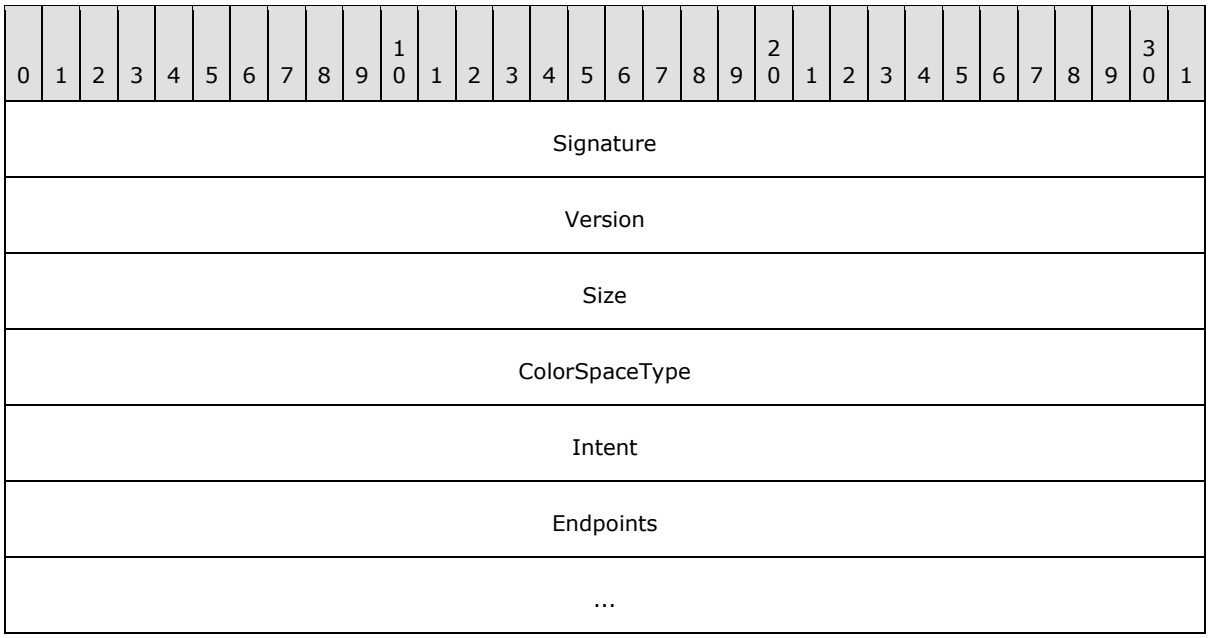
The relation between **tri-stimulus** values X,Y,Z and chromaticity values x,y,z is as follows:

```
x = X/(X+Y+Z)
y = Y/(X+Y+Z)
z = Z/(X+Y+Z)
```

If the **ColorSpaceType** field is set to LCS_sRGB or LCS_WINDOWS_COLOR_SPACE, the other fields of this object MUST be ignored, and the sRGB color space MUST be used.

2.2.2.6 LogColorSpaceW Object

The Windows Metafile Format (WMF) LogColorSpaceW Object defines information about the logical color space object with a 16-bit character file name.



...
...
...
...
...
...
...
GammaRed
GammaGreen
GammaBlue
Filename (variable)
...

Signature (4 bytes): A 32-bit unsigned integer that specifies the signature of color space objects. This MUST be set to the value 0x50534F43, which is the ASCII encoding of the string "PSOC".

Version (4 bytes): A 32-bit unsigned integer that defines the version number; it MUST be 0x00000400.

Size (4 bytes): A 32-bit unsigned integer that defines the size of this object, in bytes.

ColorSpaceType (4 bytes): A 32-bit signed integer that defines the color space type. It MAY be defined in the **LogicalColorSpace Enumeration**.

Intent (4 bytes): A 32-bit signed integer that defines the gamut mapping intent. It MAY be defined in the **GamutMappingIntent Enumeration**.

Endpoints (36 bytes): A 288-bit CIEXYZTriple Object that defines the Commission Internationale de l'Eclairage International (CIE) chromaticity x, y, and z coordinates of the three colors that correspond to the red, green, and blue endpoints for the logical color space associated with the bitmap. This field is ignored unless the **ColorSpaceType** field specifies LCS_CALIBRATED_RGB.

GammaRed (4 bytes): A 32-bit fixed point that defines the toned response curve for red. This field is ignored unless the color values are calibrated red green blue (RGB) values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

GammaGreen (4 bytes): A 32-bit fixed point that defines the toned response curve for green. This field is ignored unless the color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

GammaBlue (4 bytes): A 32-bit fixed point that defines the toned response curve for blue. This field is ignored unless the color values are calibrated RGB values and **ColorSpaceType** is set to LCS_CALIBRATED_RGB.

The gamma field format is an unsigned "8.8" fixed point integer that is then left-shifted by 8 bits. "8.8" means "8 integer bits followed by 8 fraction bits": nnnnnnnnffffffff. Taking the shift into account, the required format of the 32-bit **DWORD** is: 00000000nnnnnnnnffffffff00000000.

Filename (variable): A variable-length, 16-bit character array that names a color profile file. This field MAY be used to set the color space to be exactly as specified by the color profile. This is useful for devices that input color values for a specific printer, or when using an installable image color matcher. If a color profile is specified, all other fields of this structure SHOULD be set to reasonable values, even if the values are not completely accurate.

If the **ColorSpaceType** field is set to LCS_sRGB or LCS_WINDOWS_COLOR_SPACE, the other fields of this structure are ignored, and Windows Color System (WCS) uses the sRGB color space. The **Endpoints**, **GammaRed**, **GammaGreen**, and **GammaBlue** fields are used to describe the logical color space. The **Endpoints** field is a CIEXYZTriple Object that contains the x, y, and z values of the color space's RGB endpoint.

Whenever the **Filename** field contains a file name and the **ColorSpaceType** field is set to LCS_CALIBRATED_RGB, processing SHOULD ignore the other fields of this structure. It MUST use the color space in the file as the color space to which this LogColorSpaceW Object refers.

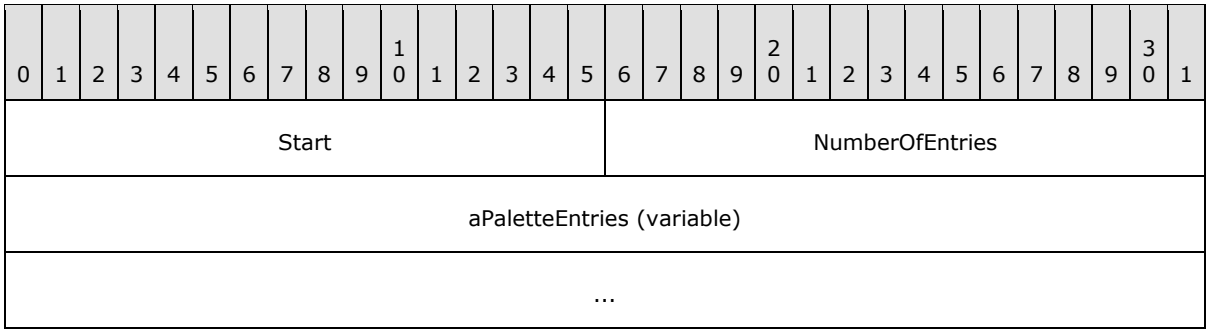
The relation between tri-stimulus values X,Y,Z and chromaticity values x,y,z is as follows:

```
x = X/(X+Y+Z)
y = Y/(X+Y+Z)
z = Z/(X+Y+Z)
```

If the **ColorSpaceType** field is set to LCS_sRGB or LCS_WINDOWS_COLOR_SPACE, the other fields of this object MUST be ignored, and the sRGB color space MUST be used.

2.2.2.7 Palette Object

The Windows Metafile Format Palette Object specifies the colors in a logical palette.



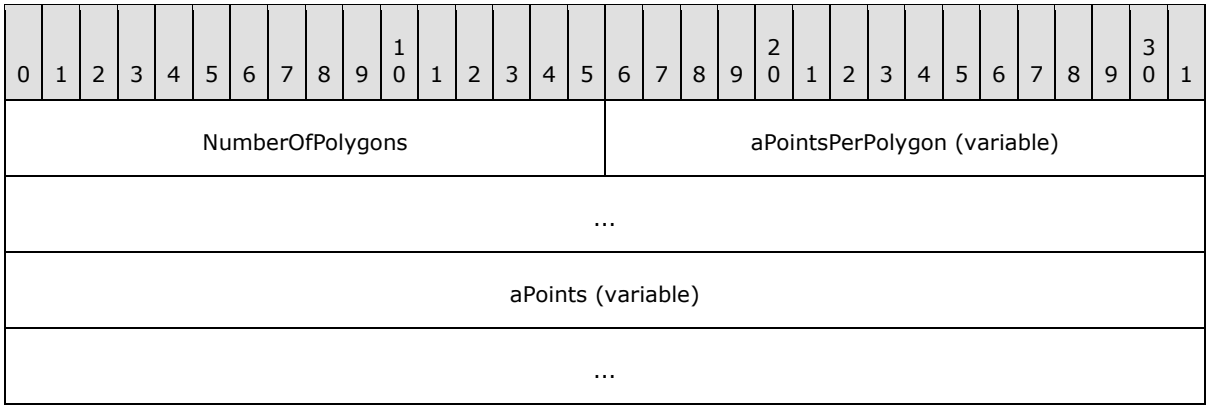
Start (2 bytes): A 16-bit unsigned integer that defines the offset into the Palette Object when used with the META_SETPALENTRIES and META_ANIMATEPALETTE record types. When used with META_CREATEPALETTE, it MUST be 0x0300.

NumberOfEntries (2 bytes): A 16-bit unsigned integer that defines the number of objects in **aPaletteEntries**.

aPaletteEntries (variable): An array of **NumberOfEntries** 32-bit PaletteEntry Objects.

2.2.2.8 PolyPolygon Object

The Windows Metafile Format PolyPolygon Object defines a series of closed polygons.



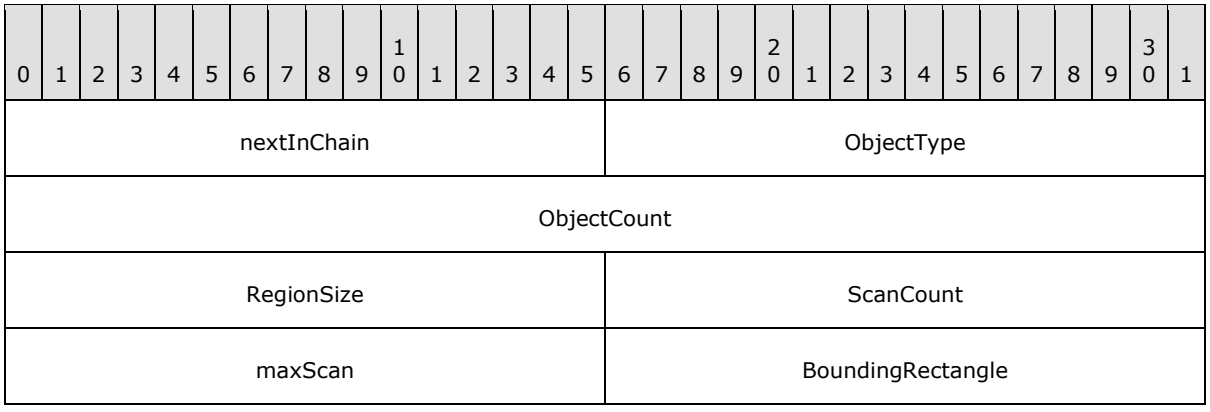
NumberOfPolygons (2 bytes): A 16-bit unsigned integer that defines the number of polygons in the object.

aPointsPerPolygon (variable): A **NumberOfPolygons** array of 16-bit unsigned integers that define the number of points for each polygon in the object.

aPoints (variable): An array of 16-bit unsigned integers that define the coordinates of the polygons.

2.2.2.9 Region Object

The Windows Metafile Format Region Object defines a potentially non-rectilinear shape defined by an array of scan lines.



...	
...	aScans (variable)
...	

nextInChain (2 bytes): A 16-bit signed integer that SHOULD NOT be used.<23>

ObjectType (2 bytes): A 16-bit signed integer that specifies the region identifier. It MUST be 0x0006.

ObjectCount (4 bytes): A 32-bit signed integer that SHOULD NOT be used.<24>

RegionSize (2 bytes): A 16-bit signed integer that defines the size of the region in bytes plus the size of **aScans** in bytes.

ScanCount (2 bytes): A 16-bit signed integer that defines the number of scans composing the region.

maxScan (2 bytes): A 16-bit signed integer that defines the maximum number of points in any one scan in this region.

BoundingRectangle (8 bytes): A 64-bit Rect Object that defines the bounding rectangle.

aScans (variable): A **ScanCount** length array of 96-bit Scan Objects that defines scan line description of the region.

2.2.2.10 Scan Object

The Windows Metafile Format Scan Object defines information about the intersection of a region with the area between two horizontal lines, or "scan lines".

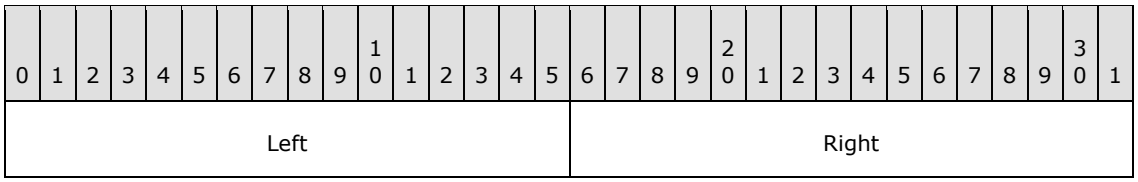
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Count																Top															
Bottom																ScanLines (variable)															
...																															
Count2																															

Count (2 bytes): A 16-bit unsigned integer that defines the scan point count, which refers to the number of horizontal scan lines specified by the **ScanLines** array.

Top (2 bytes): A 16-bit unsigned integer that defines the y-coordinate, in logical units, of the top scan line.

Bottom (2 bytes): A 16-bit unsigned integer that defines the y-coordinate, in logical units, of the bottom scan line.

ScanLines (variable): An array of scan lines, each specified by left and right x- coordinates.



Left (2 bytes): A 16-bit unsigned integer that defines the x-coordinate, in logical units, of the left end of the scan line.

Right (2 bytes): A 16-bit unsigned integer that defines the x-coordinate, in logical units, of the right end of the scan line.

Count2 (2 bytes): A 16-bit unsigned integer that MUST be the same as the value of the **Count** field; it is present to allow upward travel in the structure.

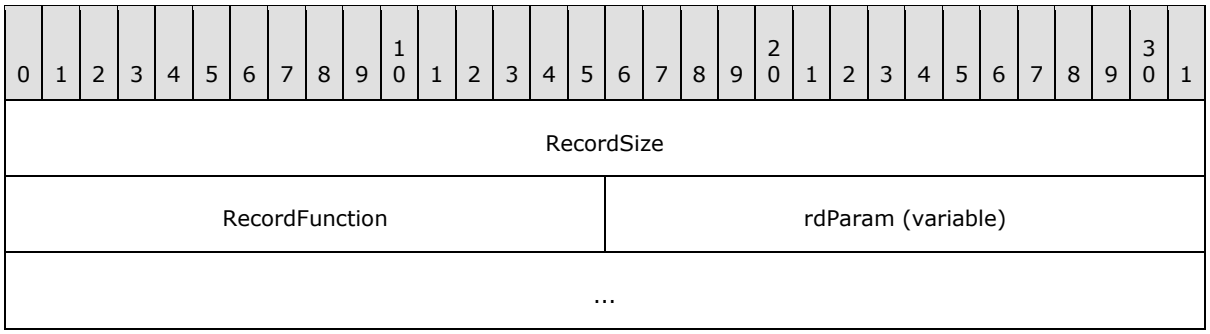
2.3 WMF Records

This section specifies the Windows Metafile Format (WMF) records, which can be grouped into the following general categories:

Name	Section	Description
Bitmap record types	2.3.1	Manage and output bitmaps.
Control record types	2.3.2	Define the start and end of a WMF metafile.
Drawing record types	2.3.3	Perform graphics drawing orders.
Object record types	2.3.4	Create and manage graphics objects.
State record types	2.3.5	Specify and manage the graphics configuration.
Escape record types	2.3.6	Specify extensions to functionality that are not directly available through other records defined in the WMF **RecordType Enumeration (section 2.1.1)**.

When a WMF metafile is processed, the order in which graphics output is performed MUST be the same as the order of drawing records in the metafile. Thus, a given drawing command is always rendered on top of the renderings of preceding commands.

The following packet definition specifies the generic structure of all WMF records except Control records.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of 16-bit **WORDs** in the record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines the type of this record. The low-order byte MUST match the low-order byte of one of the values in the **RecordType Enumeration**.

rdParam (variable): An optional place holder that is provided for record-specific fields.

2.3.1 Bitmap Record Types

This section defines the Windows Metafile Format (WMF) Bitmap Record Types, which specify records that manage and output bitmaps.

The following are the Bitmap Record Types:

Name	Section	Description
META_BITBLT	2.3.1.1	Specifies the transfer of a block of pixels from a Bitmap16 Object (section 2.2.2.1) source to a destination.
META_DIBBITBLT	2.3.1.2	Specifies the transfer of a block of pixels from a DeviceIndependentBitmap (DIB) Object (section 2.2.2.3) source to a destination rectangle.
META_DIBSTRETCHBLT	2.3.1.3	Specifies the transfer of a block of pixels from a DIB source to a destination rectangle, with possible expansion or contraction.
META_SETDIBTODEV	2.3.1.4	Defines the pixels in a specified rectangle using color data from a DIB Object.
META_STRETCHBLT	2.3.1.5	Specifies the transfer of a block of pixels from a bitmap source to a destination, with possible expansion or contraction.
META_STRETCHDIB	2.3.1.6	Renders a DIB Object, based on the color data for a specified source rectangle of pixels and a specified destination rectangle.

2.3.1.1 META_BITBLT Record

The Windows Metafile Format META_BITBLT record specifies the transfer of a block of pixels from a source to a drawing destination. A Bitmap16 Object MAY be specified as the source; a raster operation specifies how a rectangle of pixels from the source is converted into a rectangle of pixels on the destination.

There are two forms of META_BITBLT, one which contains an embedded bitmap, and another without an embedded bitmap. Many of the fields in the two varieties of META_BITBLT are the same and are defined below. The subsections which follow contain packet diagrams and specify the differences.

RecordSize: A 32-bit unsigned integer that defines the number of 16-bit **WORDs** in the WMF record.

RecordFunction: A 16-bit unsigned integer that defines this WMF record type. The lower-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_BITBLT. The high-order byte MUST contain the number of 16-bit **WORDs** of data in the record--that is, minus the first 3, **RecordSize** and **RecordFunction**.

RasterOperation: A 32-bit unsigned integer that defines the raster operation code. This code MUST be one of the values in the **Ternary Raster Operation Enumeration** table.

YSrc: A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the source rectangle.

XSrc: A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the source rectangle.

Height: A 16-bit signed integer that defines the height, in logical units, of the source and destination rectangles.

Width: A 16-bit signed integer that defines the width, in logical units, of the source and destination rectangles.

YDest: A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

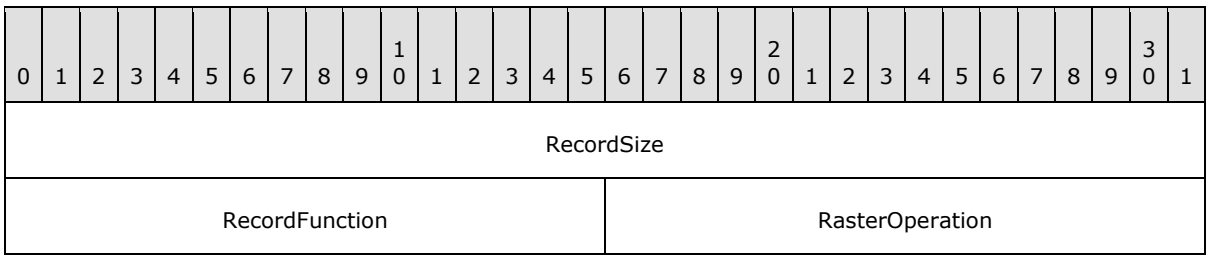
XDest: A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

The **RecordSize** and **RecordFunction** fields SHOULD be used to differentiate between the two forms of META_BITBLT. If the following Boolean expression is TRUE, there is no bitmap embedded in the record.

```
RecordSize == ((RecordFunction >> 8) + 3)
```

2.3.1.1.1 With Bitmap

This section specifies the structure of the META_BITBLT record when it contains an embedded bitmap. Fields not specified in this section are specified in the META_BITBLT section above.



...	YSrc
XSrc	Height
Width	YDest
XDest	Target (variable)
...	

Target (variable): A variable-sized Bitmap16 Object that defines source image content. This object MUST be specified, even if the raster operation does not require a source.

If the raster operation specified in this record requires a source, the embedded Bitmap16 Object MUST be used.

See section 2.3.1 for the specification of similar records.

2.3.1.1.2 Without Bitmap

This section specifies the structure of the META_BITBLT record when it does not contain an embedded source Bitmap16 Object. The source for this operation MUST be the specified rectangle on the destination surface.

Fields without descriptions in this section are specified in the META_BITBLT section above.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																YSrc															
XSrc																Reserved															
Height																Width															
YDest																XDest															

Reserved (2 bytes): This field SHOULD NOT be used.

If the raster operation specified in this record requires a source, the destination surface MUST be used; that is, the source and destination objects MUST be the same.

See section 2.3.1 for the specification of similar records.

2.3.1.2 META_DIBBITBLT Record

The Windows Metafile Format META_DIBBITBLT record specifies the transfer of a block of pixels from a source to the drawing destination. A DeviceIndependentBitmap Object MAY be specified as the source; a raster operation specifies how a rectangle of pixels from the source is converted into a rectangle of pixels on the destination.

There are two forms of META_DIBBITBLT, one which contains an embedded DIB, and another without an embedded DIB. Many of the fields in the two varieties of META_DIBBITBLT are the same and are defined below. The subsections which follow contain packet diagrams and specify the differences.

RecordSize: A 32-bit unsigned integer that defines the number of 16-bit **WORDs** in the WMF record.

RecordFunction: A 16-bit unsigned integer that defines this WMF record type. The lower-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_DIBBITBLT. The high-order byte MUST contain the number of 16-bit **WORDs** of data in the record--that is, minus the first 3, **RecordSize** and **RecordFunction**.

RasterOperation: A 32-bit unsigned integer that defines how the source pixels, the destination playback device context's current brush, and the destination pixels are to be combined to form the new image. **RasterOperation** must be one of the values in the **Ternary Raster Operation Enumeration** table.

YSrc: A 16-bit signed integer that defines the y-coordinate, in pixels, of the source rectangle in the DIB.

XSrc: A 16-bit signed integer that defines the x-coordinate, in pixels, of the source rectangle in the DIB.

Height: A 16-bit signed integer that defines the height, in logical units, of the source and destination rectangles.

Width: A 16-bit signed integer that defines the width, in logical units, of the source and destination rectangles.

YDest: A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

XDest: A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

The **RecordSize** and **RecordFunction** fields SHOULD be used to differentiate between the two forms of META_DIBBITBLT. If the following Boolean expression is TRUE, there is no bitmap embedded in the record.

```
RecordSize == ((RecordFunction >> 8) + 3)
```

2.3.1.2.1 With Bitmap

This section specifies the structure of the META_DIBBITBLT record when it contains an embedded bitmap. Fields not specified in this section are specified in the META_DIBBITBLT section above.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																YSrc															
XSrc																Height															
Width																YDest															
XDest																Target (variable)															
...																															

Target (variable): A variable-sized DeviceIndependentBitmap Object (section 2.2.2.3) that defines image content. This object MUST be specified, even if the raster operation does not require a source.

If the raster operation specified in this record requires a source, the embedded DeviceIndependentBitmap Object (section 2.2.2.3) MUST be used.

See section 2.3.1 for the specification of similar records.

2.3.1.2.2 Without Bitmap

This section specifies the structure of the META_DIBBITBLT record when it does not contain an embedded source DeviceIndependentBitmap Object. The source for this operation MUST be the specified rectangle on the destination surface.

Fields not specified in this section are specified in the META_DIBBITBLT section above.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																YSrc															
XSrc																Reserved															
Height																Width															

YDest	XDest

Reserved (2 bytes): This field SHOULD NOT be used.

If the raster operation specified in this record requires a source, the processing of this record fails.

See section 2.3.1 for the specification of similar records.

2.3.1.3 META_DIBSTRETCHBLT Record

The Windows Metafile Format META_DIBSTRETCHBLT record specifies an operation to transfer a block of pixels from a source to the drawing destination. A DeviceIndependentBitmap Object MAY be specified as the source; a raster operation and stretching mode specify how a rectangle of pixels from the source is converted into pixels on the destination.

The DIB is stretched or compressed according to the stretching mode currently set in the destination playback device context.

There are two forms of META_DIBSTRETCHBLT, one which contains an embedded DIB, and another without an embedded DIB. Many of the fields in the two varieties of META_DIBSTRETCHBLT are the same and are defined below. The subsections which follow contain packet diagrams and specify the differences.

RecordSize: A 32-bit unsigned integer that defines the number of 16-bit **WORDs** in the WMF record.

RecordFunction: A 16-bit unsigned integer that defines this WMF record type. The lower-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_DIBSTRETCHBLT. The high-order byte MUST contain the number of 16-bit **WORDs** of data in the record--that is, minus the first 3, **RecordSize** and **RecordFunction**.

RasterOperation: A 32-bit unsigned integer that defines how the source pixels, the destination playback device context's current brush, and the destination pixels are to be combined to form the new image. **RasterOperation** must be one of the values in the **Ternary Raster Operation Enumeration** table.

SrcHeight: A 16-bit signed integer that defines the height, in logical units, of the source rectangle in the DIB Object.

SrcWidth: A 16-bit signed integer that defines the width, in logical units, of the source rectangle in the DIB Object.

YSrc: A 16-bit signed integer that defines the y-coordinate, in pixels, of the source rectangle in the DIB Object.

XSrc: A 16-bit signed integer that defines the x-coordinate, in pixels, of the source rectangle in the DIB Object.

DestHeight: A 16-bit signed integer that defines the height, in logical units, of the destination rectangle in the DIB Object.

DestWidth: A 16-bit signed integer that defines the width, in logical units, of the destination rectangle in the DIB Object.

YDest: A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

XDest: A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

The **RecordSize** and **RecordFunction** fields SHOULD be used to differentiate between the two forms of META_DIBSTRETCHBLT. If the following Boolean expression is TRUE, there is no bitmap embedded in the record.

```
RecordSize == ((RecordFunction >> 8) + 3)
```

2.3.1.3.1 With Bitmap

This section specifies the structure of the META_DIBSTRETCHBLT record when it contains an embedded bitmap. Fields not specified in this section are specified in the META_DIBSTRETCHBLT section above.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">RecordSize</td></tr>
<tr><td colspan="16">RecordFunction</td><td colspan="16">RasterOperation</td></tr>
<tr><td colspan="16">...</td><td colspan="16">SrcHeight</td></tr>
<tr><td colspan="16">SrcWidth</td><td colspan="16">YSrc</td></tr>
<tr><td colspan="16">XSrc</td><td colspan="16">DestHeight</td></tr>
<tr><td colspan="16">DestWidth</td><td colspan="16">YDest</td></tr>
<tr><td colspan="16">XDest</td><td colspan="16">Target (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

Target (variable): A variable-sized DIB Object (section 2.2.2.3) that defines image content. This object MUST be specified, even if the raster operation does not require a source.

If the raster operation specified in this record requires a source, the embedded DeviceIndependentBitmap Object MUST be used.

See section 2.3.1 for the specification of similar records.

2.3.1.3.2 Without Bitmap

This section specifies the structure of the META_DIBSTRETCHBLT record when it does not contain an embedded source DeviceIndependentBitmap Object. The source for this operation MUST be the specified rectangle on the destination surface.

Fields not specified in this section are specified in the META_DIBSTRETCHBLT section above.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																YSrc															
XSrc																Reserved															
DestHeight																DestWidth															
YDest																XDest															

Reserved (2 bytes): This field SHOULD NOT be used.

If the raster operation specified in this record requires a source, the processing of this record fails.

See section 2.3.1 for the specification of similar records.

2.3.1.4 META_SETDIBTODEV Record

The Windows Metafile Format META_SETDIBTODEV record defines the pixels in the specified rectangle on the device that is associated with the destination playback device context by using color data from a DeviceIndependentBitmap Object.

META_SETDIBTODEV has been extended to allow a JPEG or PNG image to be passed as the source image.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																ColorUsage															
ScanCount																StartScan															
yDib																xDib															
Height																Width															
yDest																xDest															

DIB (variable)
...

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the Windows Metafile Format record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this Windows Metafile Format record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETDIBTODEV.

ColorUsage (2 bytes): A 16-bit unsigned integer that defines whether the **Colors** field of the DeviceIndependentBitmap Object in the **DIB** contains explicit red, green, blue (RGB) values or indexes into a palette. The **ColorUsage** MUST be one of the values in the **ColorUsage Enumeration** table.

ScanCount (2 bytes): A 16-bit unsigned integer that defines the number of DIB scan lines contained in the array pointed to by the **DIB** member.

StartScan (2 bytes): A 16-bit unsigned integer that defines the starting scan line in the DIB.

yDib (2 bytes): A 16-bit unsigned integer that defines the y-coordinate, in logical units, of the lower-left corner of the DIB.

xDib (2 bytes): A 16-bit unsigned integer that defines the x-coordinate, in logical units, of the lower-left corner of the DIB.

Height (2 bytes): A 16-bit unsigned integer that defines height, in logical units, of the DIB.

Width (2 bytes): A 16-bit unsigned integer that defines width, in logical units, of the DIB.

yDest (2 bytes): A 16-bit unsigned integer that defines the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

xDest (2 bytes): A 16-bit unsigned integer that defines the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

DIB (variable): A variable-length DeviceIndependentBitmap Object that defines image content.

See section 2.3.1 for the specification of similar records.

2.3.1.5 META_STRETCHBLT Record

The Windows Metafile Format META_STRETCHBLT record specifies an operation to transfer a block of pixels from a source to the drawing destination. A Bitmap16 Object MAY be specified as the source; a raster operation specifies how a rectangle of pixels from the source is converted into a rectangle of pixels on the destination.

There are two forms of META_STRETCHBLT, one which contains an embedded bitmap, and another without an embedded bitmap. Many of the fields in the two varieties of META_STRETCHBLT are the same and are defined below. The subsections which follow contain packet diagrams and specify the differences.

RecordSize: A 32-bit unsigned integer that defines the number of 16-bit **WORDs** in the WMF record.

RecordFunction: A 16-bit unsigned integer that defines this WMF record type. The lower-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_STRETCHBLT. The high-order byte MUST contain the number of 16-bit **WORDs** of data in the record--that is, minus the first 3, **RecordSize** and **RecordFunction**.

RasterOperation: A 32-bit unsigned integer that defines the raster operation code. This code MUST be one of the values in the **Ternary Raster Operation Enumeration** table.

SrcHeight: A 16-bit signed integer that defines the height, in logical units, of the source rectangle.

SrcWidth: A 16-bit signed integer that defines the width, in logical units, of the source rectangle.

YSrc: A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the source rectangle.

XSrc: A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the source rectangle.

DestHeight: A 16-bit signed integer that defines the height, in logical units, of the source and destination rectangles.

DestWidth: A 16-bit signed integer that defines the width, in logical units, of the source and destination rectangles.

YDest: A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

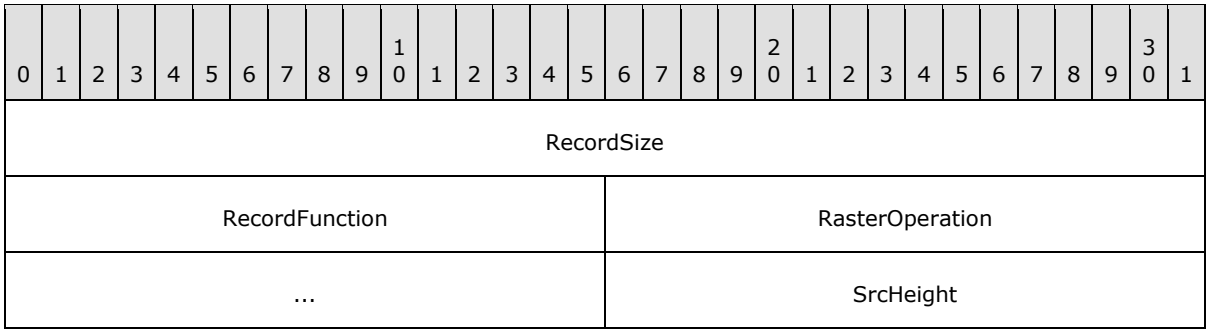
XDest: A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

The **RecordSize** and **RecordFunction** fields SHOULD be used to differentiate between the two forms of META_STRETCHBLT. If the following Boolean expression is TRUE, there is no bitmap embedded in the record.

```
RecordSize == ((RecordFunction >> 8) + 3)
```

2.3.1.5.1 With Bitmap

This section specifies the structure of the META_STRETCHBLT record when it contains an embedded bitmap. Fields not specified in this section are specified in the META_STRETCHBLT section above.



SrcWidth	YSrc
XSrc	DestHeight
DestWidth	YDest
XDest	Target (variable)
...	

Target (variable): A variable-sized Bitmap16 Object that defines source image content. This object MUST be specified, even if the raster operation does not require a source.

If the raster operation specified in this record requires a source, the embedded Bitmap16 Object MUST be used.

See section 2.3.1 for the specification of similar records.

2.3.1.5.2 Without Bitmap

This section specifies the structure of the META_STRETCHBLT record when it does not contain an embedded source Bitmap16 Object. The source for this operation MUST be the specified rectangle on the destination surface.

Fields not specified in this section are specified in the META_STRETCHBLT section above.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																SrcHeight															
SrcWidth																YSrc															
XSrc																Reserved															
DestHeight																DestWidth															
YDest																XDest															

Reserved (2 bytes): This field SHOULD NOT be used.

If the raster operation specified in this record requires a source, the destination surface MUST be used; that is, the source and destination objects MUST be the same.

See section 2.3.1 for the specification of similar records.

2.3.1.6 META_STRETCHDIB Record

The Windows Metafile Format META_STRETCHDIB record renders a DeviceIndependentBitmap Object, based on the color data for a specified source rectangle of pixels and a specified destination rectangle.

If the destination rectangle is larger than the source rectangle, this function stretches the rows and columns of color data to fit the destination rectangle. If the destination rectangle is smaller than the source rectangle, this function compresses the rows and columns by using the specified raster operation.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																ColorUsage															
SrcHeight																SrcWidth															
YSrc																XSrc															
DestHeight																DestWidth															
yDst																xDst															
DIB (variable)																															
...																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_STRETCHDIB.

RasterOperation (4 bytes): A 32-bit unsigned integer that defines how the source pixels, the destination playback device context's current brush, and the destination pixels are to be combined to form the new image. **RasterOperation** MUST be one of the values in the **Ternary Raster Operation Enumeration** table.

ColorUsage (2 bytes): A 16-bit unsigned integer that defines whether the **Colors** field of **DIB** contains explicit red, green, blue (RGB) values or indexes into a palette. **ColorUsage** MUST be one of the values in the **ColorUsage Enumeration** table.

SrcHeight (2 bytes): : A 16-bit signed integer that defines the height, in pixels, of the source rectangle in the **DIB**.

SrcWidth (2 bytes): : A 16-bit signed integer that defines the width, in pixels, of the source rectangle in the **DIB**.

YSrc (2 bytes): A 16-bit signed integer that defines the y-coordinate, in pixels, of the source rectangle in the **DIB**.

XSrc (2 bytes): A 16-bit signed integer that defines the x-coordinate, in pixels, of the source rectangle in the **DIB**.

DestHeight (2 bytes): A 16-bit signed integer that defines the height, in logical units, of the destination rectangle.

DestWidth (2 bytes): A 16-bit signed integer that defines the width, in logical units, of the destination rectangle.

yDst (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

xDst (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

DIB (variable): A variable-sized DeviceIndependentBitmap Object that defines the image content.

See section 2.3.1 for the specification of similar records.

2.3.2 Control Record Types

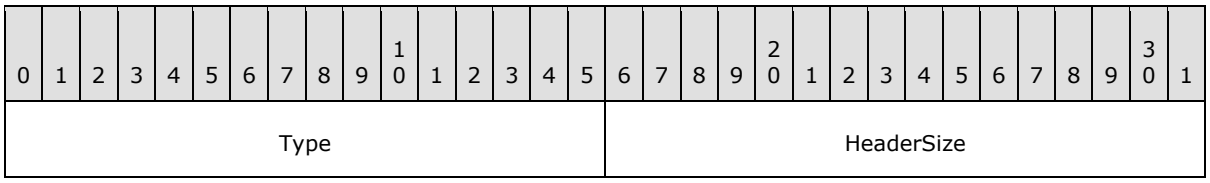
This section defines the Windows Metafile Format (WMF) Control Record Types, which specify records that begin and end a WMF metafile.

The following are the Control Record Types:

Name	Section	Description
META_HEADER	2.3.2.1	Specifies the start of a WMF metafile.
META_EOF	2.3.2.2	Specifies the end of a WMF metafile.

2.3.2.1 META_HEADER Record

The Windows Metafile Format META_HEADER record is the first record in a standard WMF metafile.



Version	SizeLow
SizeHigh	NumberOfObjects
MaxRecord	
NumberOfMembers	

Type (2 bytes): A 16-bit unsigned integer that defines the type of metafile. It MUST be a value in the **MetafileType Enumeration** table.

HeaderSize (2 bytes): A 16-bit unsigned integer that defines the number of 16-bit words in the header.

Version (2 bytes): A 16-bit unsigned integer that defines the metafile version. It MUST be a value in the **MetafileVersion Enumeration** table.

SizeLow (2 bytes): A 16-bit unsigned integer that defines the low-order word of the number of 16-bit words in the entire metafile.

SizeHigh (2 bytes): A 16-bit unsigned integer that defines the high-order word of the number of 16-bit words in the entire metafile.

NumberOfObjects (2 bytes): A 16-bit unsigned integer that specifies the number of graphics objects that are defined in the entire metafile. These objects include brushes, pens and all the other fixed-length and variable-length objects specified in sections 2.2.1 and 2.2.2 .

MaxRecord (4 bytes): A 32-bit unsigned integer that specifies the size of the largest record used in the metafile (in 16-bit elements).

NumberOfMembers (2 bytes): A 16-bit unsigned integer that is not used. It SHOULD be 0x0000.

See section 2.3.2 for the specification of similar records.

2.3.2.2 META_EOF Record

The Windows Metafile Format META_EOF record indicates the end of the WMF metafile.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of 16-bit **WORDs** in the record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines the type of this record. For META_EOF, this value MUST be 0x0000, as specified in the **RecordType Enumeration** table.

See section 2.3.2 for the specification of similar records.

2.3.3 Drawing Record Types

This section defines the Windows Metafile Format (WMF) Drawing Record Types, which specify records that perform graphics output.

The following are the Drawing Record Types:

Name	Section	Description
META_ARC	2.3.3.1	Draws an elliptical arc.
META_CHORD	2.3.3.2	Draws a chord.
META_ELLIPSE	2.3.3.3	Draws an ellipse.
META_EXTFLOODFILL	2.3.3.4	Fills an area with the current brush.
META_EXTTEXTOUT	2.3.3.5	Outputs a character string with optional opaquing and clipping.
META_FILLREGION	2.3.3.6	Fills a region using a specified brush.
META_FLOODFILL	2.3.3.7	fills an area of the output surface with the current brush.
META_FRAMEREGION	2.3.3.8	Draws a border around a specified region using a specified brush.
META_INVERTREGION	2.3.3.9	Draws a region in which the colors are inverted.
META_LINETO	2.3.3.10	Draws a line from the current drawing position up to, but not including, the specified position.
META_PAINTREGION	2.3.3.11	Paints the specified region using the currently selected brush.
META_PATBLT	2.3.3.12	Paints a specified rectangle using the currently selected brush.
META_PIE	2.3.3.13	Draws a pie-shaped wedge bounded by the intersection of an ellipse and two radial lines.
META_POLYLINE	2.3.3.14	Draws a series of line segments by connecting the points in the specified array.
META_POLYGON	2.3.3.15	Paints a polygon consisting of two or more vertices connected by straight lines.
META_POLYPOLYGON	2.3.3.16	Paints a series of closed polygons.
META_RECTANGLE	2.3.3.17	Paints a rectangle.
META_ROUNDRECT	2.3.3.18	Paints a rectangle with rounded corners.
META_SETPIXEL	2.3.3.19	Sets the current pixel at the specified coordinates to the specified color.
META_TEXTOUT	2.3.3.20	Outputs a character string.

2.3.3.1 META_ARC Record

The Windows Metafile Format META_ARC record draws an elliptical arc.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																YEndArc															
XEndArc																YStartArc															
XStartArc																BottomRect															
RightRect																TopRect															
LeftRect																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ARC.

YEndArc (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the ending point of the radial line defining the ending point of the arc.

XEndArc (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the ending point of the radial line defining the ending point of the arc.

YStartArc (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the ending point of the radial line defining the starting point of the arc.

XStartArc (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the ending point of the radial line defining the starting point of the arc.

BottomRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

RightRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

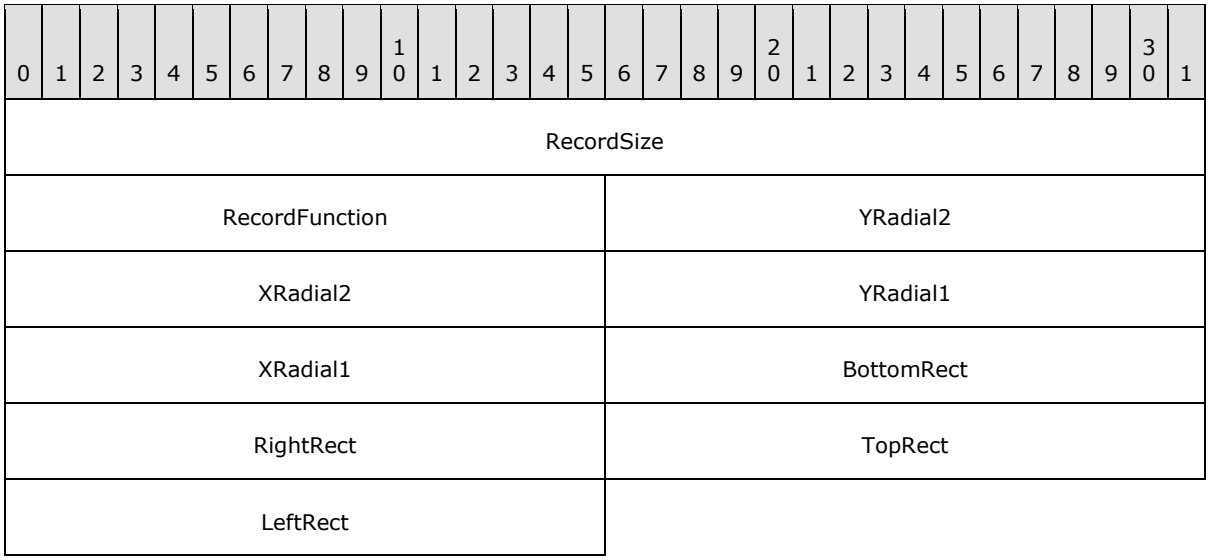
TopRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

LeftRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.2 META_CHORD Record

The Windows Metafile Format META_CHORD record draws a chord, which is defined by a region bounded by the intersection of an ellipse with a line segment. The chord is outlined by using the current pen and filled by using the current brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_CHORD.

YRadial2 (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical coordinates, of the endpoint of the second radial.

XRadial2 (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical coordinates, of the endpoint of the second radial.

YRadial1 (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical coordinates, of the endpoint of the first radial.

XRadial1 (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical coordinates, of the endpoint of the first radial.

BottomRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

RightRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

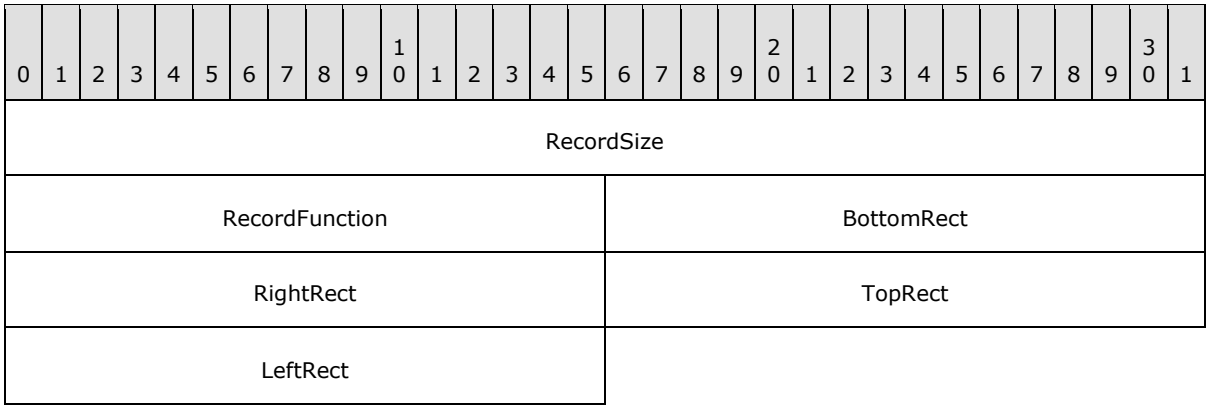
TopRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

LeftRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.3 META_ELLIPSE Record

The Windows Metafile Format META_ELLIPSE record draws an ellipse. The center of the ellipse is the center of the specified bounding rectangle. The ellipse is outlined by using the current pen and is filled by using the current brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ELLIPSE.

BottomRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

RightRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

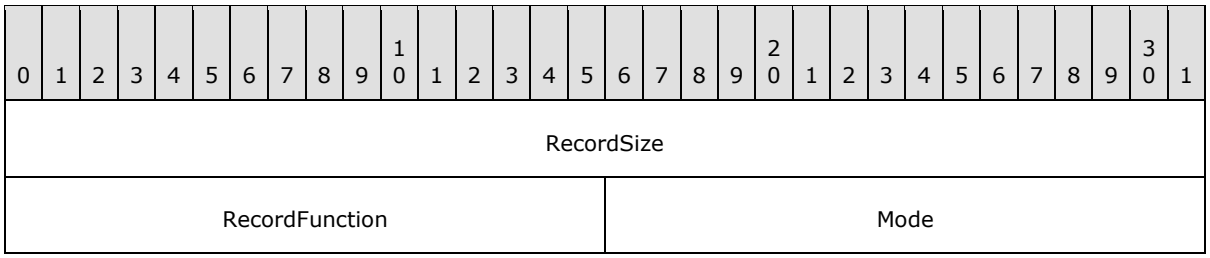
TopRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

LeftRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.4 META_EXTFLOODFILL Record

The Windows Metafile Format META_EXTFLOODFILL record fills an area with the current brush.



ColorRef	
Y	X

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_EXTFLOODFILL.

Mode (2 bytes): A 16-bit unsigned integer that defines the fill operation to be performed. This member must be one of the values in the **FloodFill Enumeration** table.

ColorRef (4 bytes): A 32-bit ColorRef Object that defines the color value.

Y (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the point to be set.

X (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the point to be set.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.5 META_EXTTEXTOUT Record

The Windows Metafile Format (WMF) META_EXTTEXTOUT record outputs text by using the currently selected font, background color, and text color. Optionally, dimensions can be provided for clipping, opaquing, or both.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																Y															
X																StringLength															
fwOpts																Rectangle (optional)															
...																															
...																String (variable)															
...																															
Dx (variable)																															

...

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_EXTTEXTOUT.

Y (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, where the text string is to be located.

X (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, where the text string is to be located.

StringLength (2 bytes): A 16-bit signed integer that defines the length of the string.

fwOpts (2 bytes): A 16-bit unsigned integer that defines the use of the application-defined rectangle. This member can be a combination of one or more values in the **ExtTextOutOptions Enumeration**.

Rectangle (8 bytes): An optional 8-byte Rect Object that defines the dimensions, in logical coordinates, of a rectangle that is used for clipping, opaquing, or both.

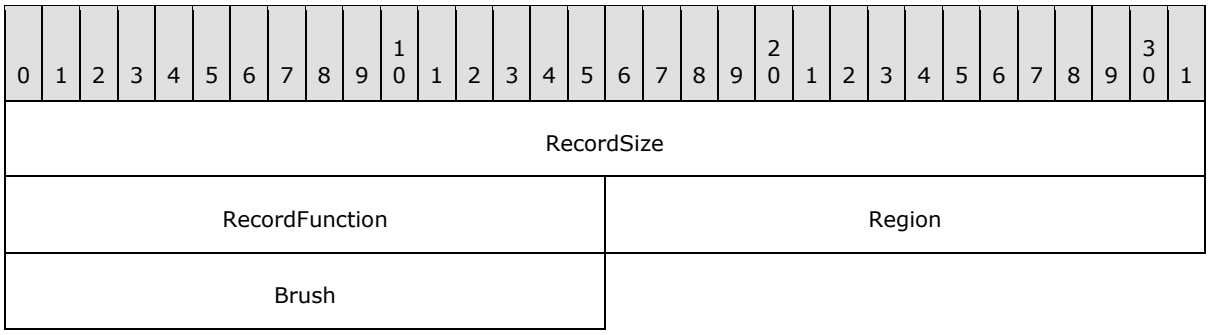
String (variable): A variable-length string that specifies the text to be drawn. The string does not need to be zero-terminated, since **StringLength** specifies the length of the string. If the length is odd, an extra byte is placed after it so that the following member (optional **Dx**) is aligned on a 16-bit boundary.

Dx (variable): An optional array of 16-bit signed integers that indicate the distance between origins of adjacent character cells. For example, **Dx**[i] logical units separate the origins of character cell i and character cell i + 1. If this field is present, there MUST be the same number of values as there are characters in the string.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.6 META_FILLREGION Record

The Windows Metafile Format META_FILLREGION record fills a region using a specified brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_FILLREGION.

Region (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the region to be filled.

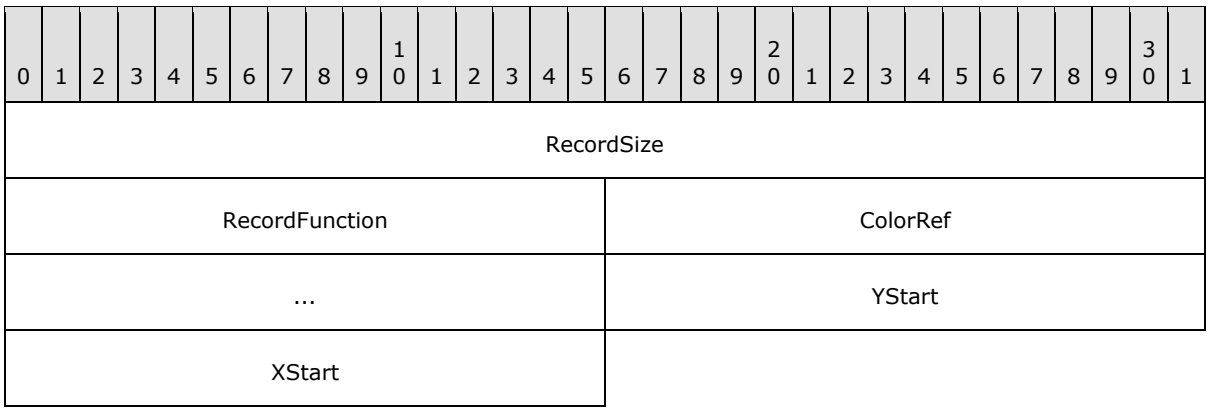
Brush (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the brush to use for filling the region.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.7 META_FLOODFILL Record

The Windows Metafile Format META_FLOODFILL record fills an area of the output surface with the current brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_FLOODFILL.

ColorRef (4 bytes): A 32-bit ColorRef Object that defines the color value.

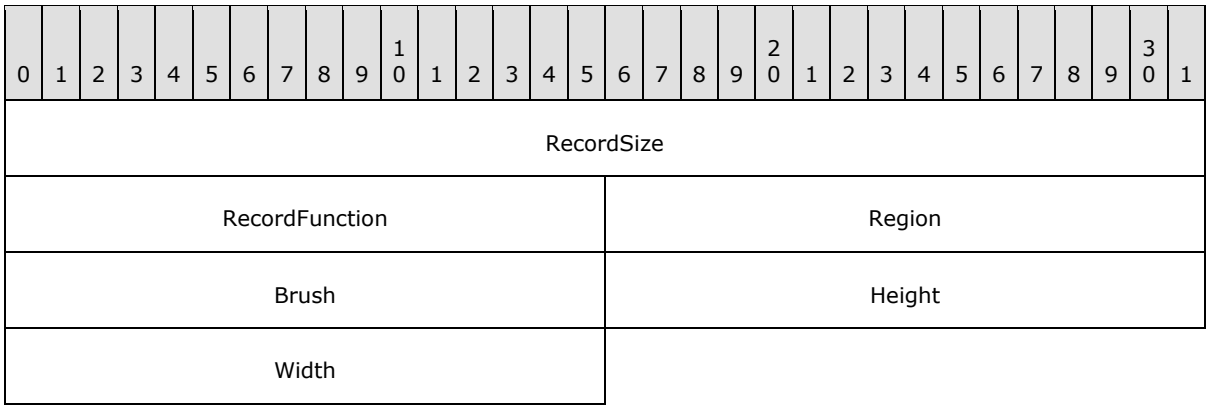
YStart (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the point where filling is to start.

XStart (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the point where filling is to start.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.8 META_FRAMEREGION Record

The Windows Metafile Format META_FRAMEREGION record draws a border around a specified region using a specified brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_FRAMEREGION.

Region (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the region to be framed.

Brush (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the Brush to use for filling the region.

Height (2 bytes): A 16-bit signed integer that defines the height, in logical units, of the region frame.

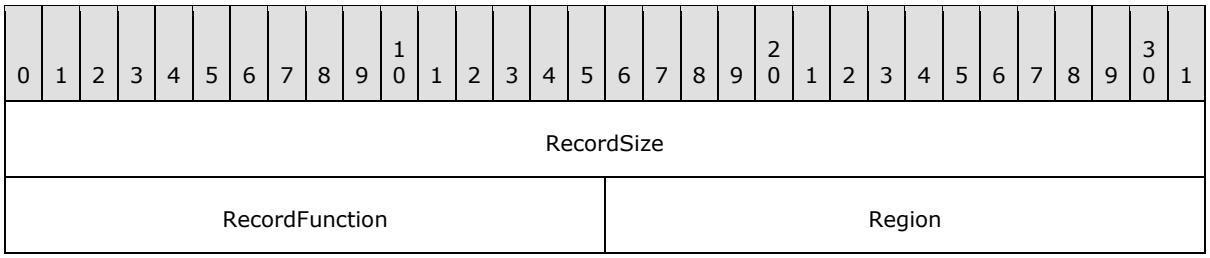
Width (2 bytes): A 16-bit signed integer that defines the width, in logical units, of the region frame.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.9 META_INVERTREGION Record

The Windows Metafile Format META_INVERTREGION record draws a region in which the colors are inverted.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_INVERTREGION.

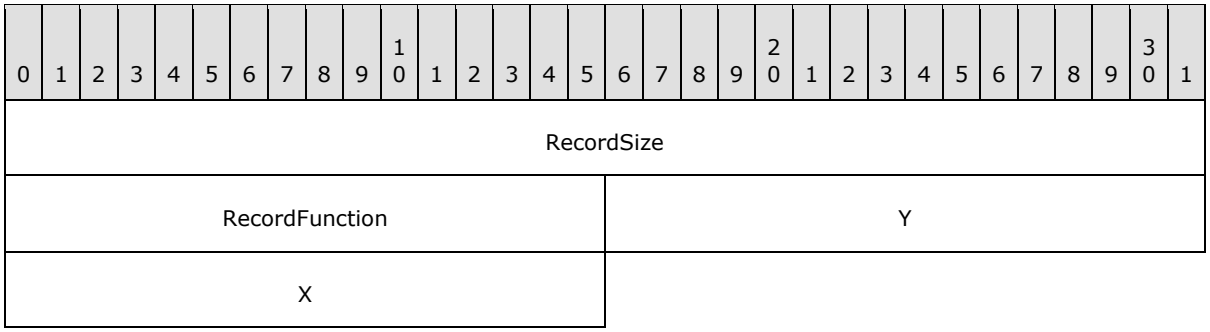
Region (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the region to be inverted.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.10 META_LINETO Record

The Windows Metafile Format META_LINETO record draws a line from the current position up to, but not including, the specified point.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_LINETO.

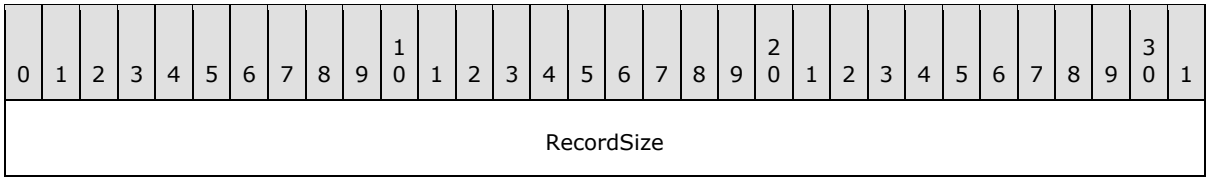
Y (2 bytes): A 16-bit signed integer that defines the vertical component of the drawing destination position, in logical units.

X (2 bytes): A 16-bit signed integer that defines the horizontal component of the drawing destination position, in logical units.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.11 META_PAINTREGION Record

The Windows Metafile Format META_PAINTREGION record paints the specified region using the currently selected brush.



RecordFunction	Region

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_PAINTREGION.

Region (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the region to be inverted.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.12 META_PATBLT Record

The Windows Metafile Format META_PATBLT record paints a specified rectangle using the currently selected brush. The brush color and the surface color or colors are combined by using the specified raster operation.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																RasterOperation															
...																Height															
Width																YLeft															
XLeft																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_PATBLT.

RasterOperation (4 bytes): A 32-bit unsigned integer that defines the raster operation code. This code MUST be one of the values in the **Ternary Raster Operation** enumeration table.

Height (2 bytes): A 16-bit signed integer that defines the height, in logical units, of the rectangle.

Width (2 bytes): A 16-bit signed integer that defines the width, in logical units, of the rectangle.

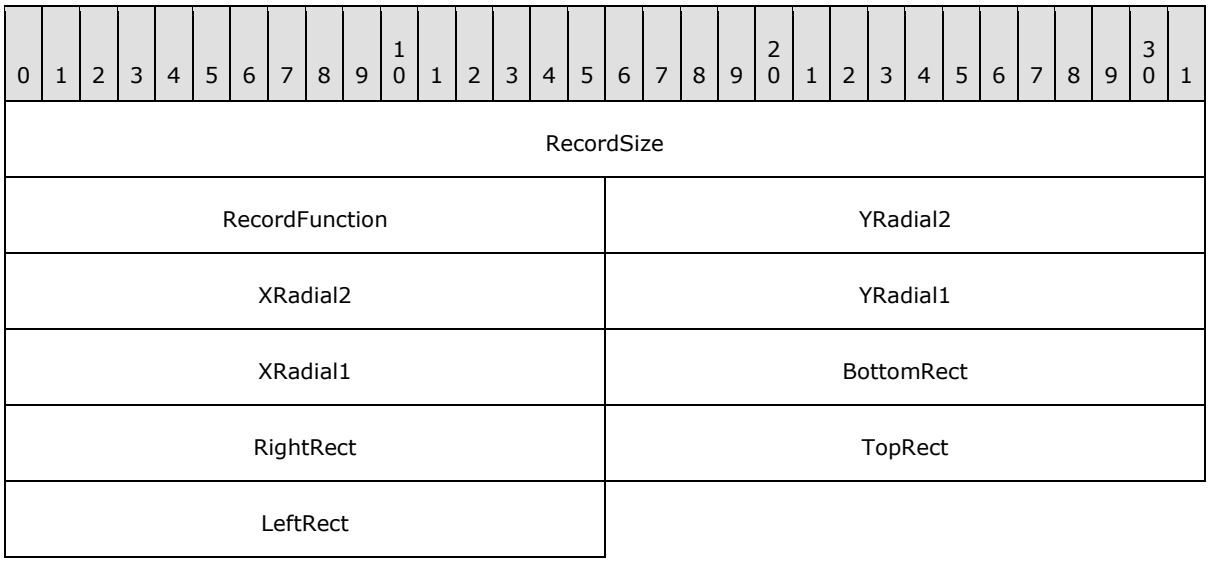
YLeft (2 bytes): A 16-bit signed integer that defines the y- coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

XLeft (2 bytes): A 16-bit signed integer that defines the x- coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.13 META_PIE Record

The Windows Metafile Format META_PIE record draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The pie is outlined by using the current pen and filled by using the current brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_PIE.

YRadial2 (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical coordinates, of the endpoint of the second radial.

XRadial2 (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical coordinates, of the endpoint of the second radial.

YRadial1 (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical coordinates, of the endpoint of the first radial.

XRadial1 (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical coordinates, of the endpoint of the first radial.

BottomRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

RightRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the bounding rectangle.

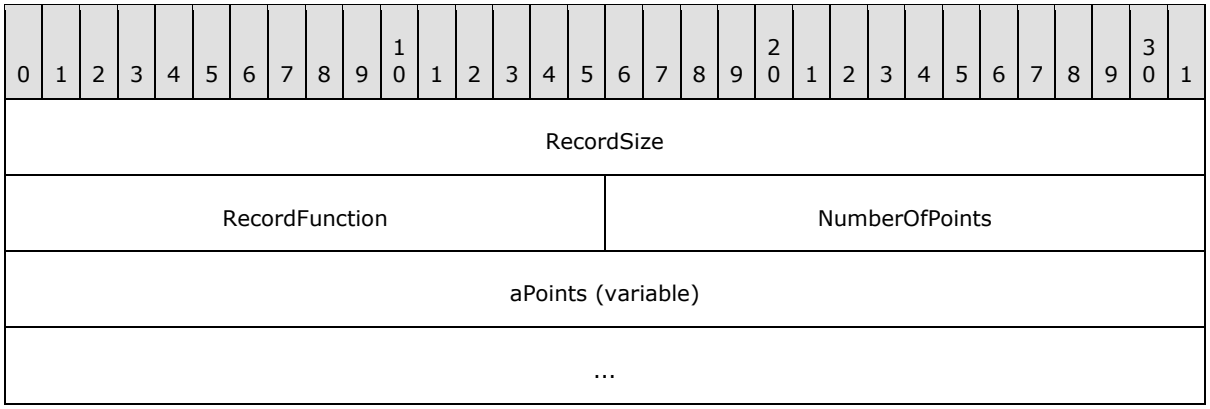
TopRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

LeftRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the bounding rectangle.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.14 META_POLYLINE Record

The Windows Metafile Format META_POLYLINE record draws a series of line segments by connecting the points in the specified array.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_POLYLINE.

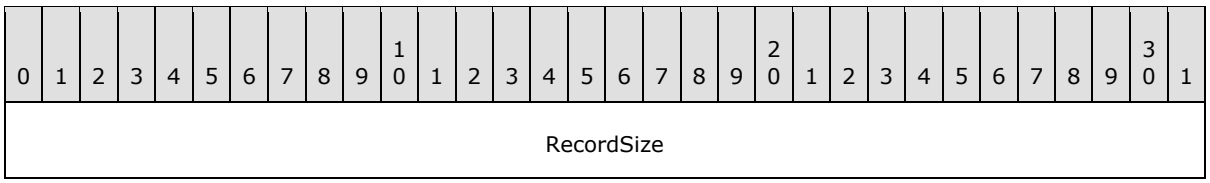
NumberOfPoints (2 bytes): A 16-bit signed integer that defines the number of points in the array.

aPoints (variable): A **NumberOfPoints** array of 32-bit PointS Objects, in logical units.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.15 META_POLYGON Record

The Windows Metafile Format META_POLYGON record paints a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.



RecordFunction	NumberofPoints
aPoints (variable)	
...	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_POLYGON.

NumberofPoints (2 bytes): A 16-bit signed integer that defines the number of points in the array.

aPoints (variable): A **NumberOfPoints** array of 32-bit PointS Objects, in logical units.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.16 META_POLYPOLYGON Record

The Windows Metafile Format META_POLYPOLYGON record paints a series of closed polygons. Each polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode. The polygons drawn by this function can overlap.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																PolyPolygon (variable)															
...																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the Windows Metafile Format record.

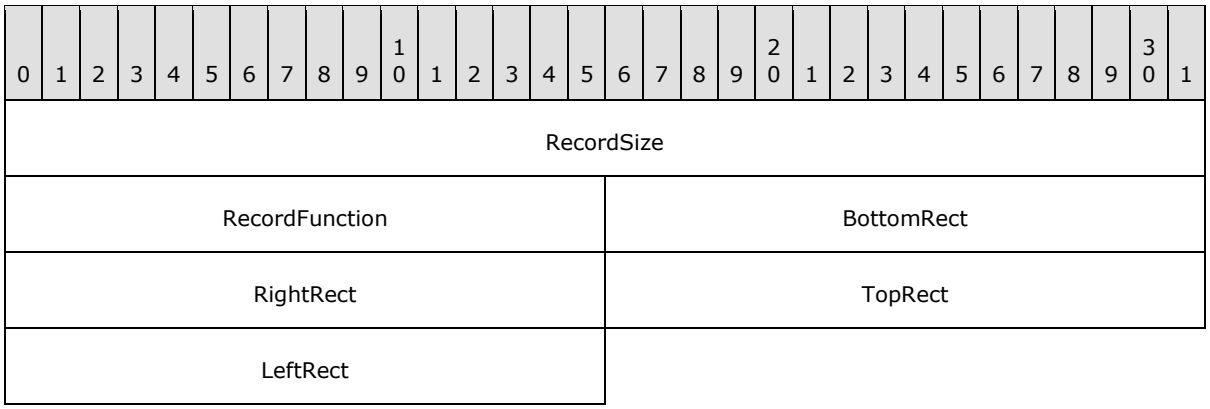
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this Windows Metafile Format record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_POLYPOLYGON.

PolyPolygon (variable): A variable-sized PolyPolygon Object that defines the point information.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.17 META_RECTANGLE Record

The Windows Metafile Format META_RECTANGLE record paints a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_RECTANGLE.

BottomRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the rectangle.

RightRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the rectangle.

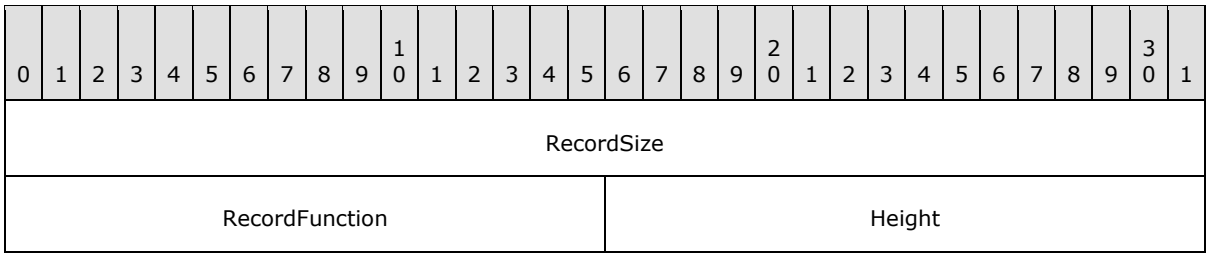
TopRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the rectangle.

LeftRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.18 META_ROUNDRECT Record

The Windows Metafile Format META_ROUNDRECT record paints a rectangle with rounded corners. The rectangle is outlined by using the current pen and filled by using the current brush.



Width	BottomRect
RightRect	TopRect
LeftRect	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ROUNDRECT.

Height (2 bytes): A 16-bit signed integer that defines the height, in logical coordinates, of the ellipse used to draw the rounded corners.

Width (2 bytes): A 16-bit signed integer that defines the width, in logical coordinates, of the ellipse used to draw the rounded corners.

BottomRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the rectangle.

RightRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the rectangle.

TopRect (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the rectangle.

LeftRect (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.19 META_SETPIXEL Record

The Windows Metafile Format META_SETPIXEL record sets the current pixel at the specified coordinates to the specified color.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																ColorRef															
...																Y															
X																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETPIXEL.

ColorRef (4 bytes): A 32-bit ColorRef Object that defines the color value.

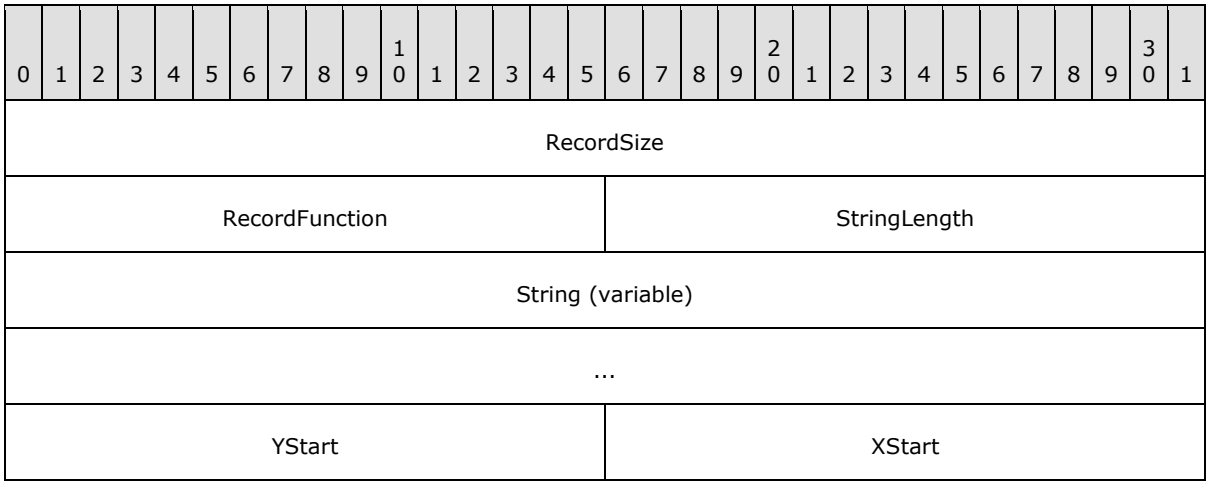
Y (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the point to be set.

X (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the point to be set.

See section 2.3.3 for the specification of other Drawing records.

2.3.3.20 META_TEXTOUT Record

The Windows Metafile Format (WMF) META_TEXTOUT record outputs a character string at the specified location, by using the currently selected font, background color, and text color.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_TEXTOUT.

StringLength (2 bytes): A 16-bit signed integer that defines the length of the string, in bytes, pointed to by **String**.

String (variable): A **StringLength** array of ANSI Character Set characters that defines the text to be drawn. The string does not need to be null-terminated, since **StringLength** specifies the length of the string. The string is written at the location specified by the **XStart** and **YStart** fields, using the currently selected font, background color, and text color.

YStart (2 bytes): A 16-bit signed integer that defines the vertical (y-axis) coordinate, in logical units, of the point where drawing is to start.

XStart (2 bytes): A 16-bit signed integer that defines the horizontal (x-axis) coordinate, in logical units, of the point where drawing is to start.

See section 2.3.3 for the specification of other Drawing records.

2.3.4 Object Record Types

This section defines the Windows Metafile Format (WMF) Object Record Types, which create and manage graphics objects.

The following are the Object Record Types:

Name	Section	Description
META_CREATEBRUSHINDIRECT	2.3.4.1	Creates a Brush Object (section 2.2.2.2) from a LogBrush Object (section 2.2.1.8).
META_CREATEFONTINDIRECT	2.3.4.2	Creates a Font Object (section 2.2.2.4).
META_CREATEPALETTE	2.3.4.3	Creates a Palette Object (section 2.2.2.7).
META_CREATEPATTERNBRUSH	2.3.4.4	Creates a Brush Object with a pattern specified by a bitmap.
META_CREATEPENINDIRECT	2.3.4.5	Creates a Pen Object (section 2.2.1.10).
META_CREATEREGION	2.3.4.6	Creates a Region Object (section 2.2.2.9).
META_DELETEOBJECT	2.3.4.7	Deletes an existing object.
META_DIBCREATEPATTERNBRUSH	2.3.4.8	Creates a Brush Object with a pattern specified by a device-independent bitmap (DIB).
META_SELECTCLIPREGION	2.3.4.9	Defines the current clipping region with a specified Region Object.
META_SELECTOBJECT	2.3.4.10	Specifies a graphics object for the playback device context.
META_SELECTPALETTE	2.3.4.11	Defines the current palette with a specified Palette Object.

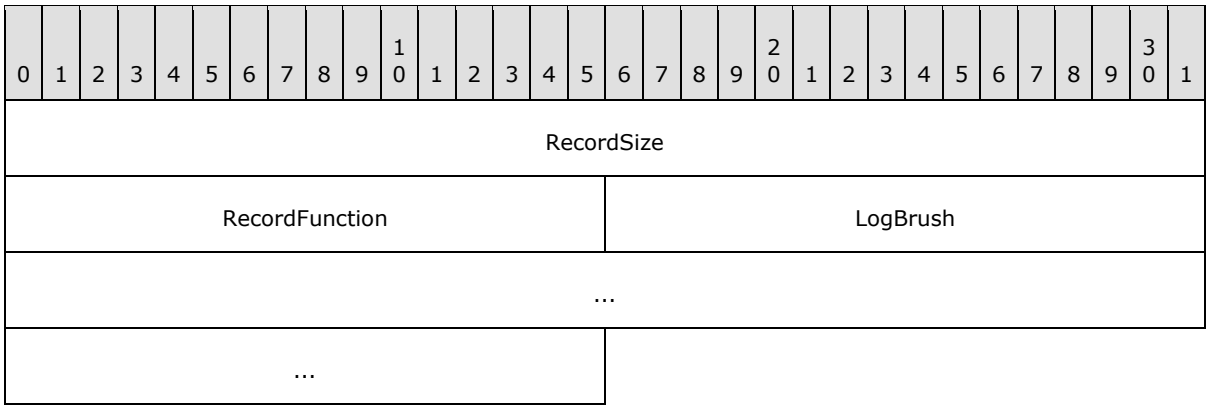
Whenever a graphics object is created by one of the metafile records above, the following actions are implied:

- The object MUST always be assigned the lowest-numbered available index in the WMF Object Table.
- Subsequent WMF records MUST refer to the object by its assigned WMF Object Table index.
- The object MUST NOT be used in drawing operations until a META_SELECTOBJECT record is received that specifies the object's index.
- The object MUST remain available for selection until a META_DELETEOBJECT (section 2.3.4.7) record is received that specifies the object's index.

For further information concerning object indexes and the WMF Object Table, see Managing Objects in section 3.1.4.

2.3.4.1 META_CREATEBRUSHINDIRECT Record

The Windows Metafile Format (WMF) META_CREATEBRUSHINDIRECT record creates a Brush Object (section 2.2.2.2) from a LogBrush Object (section 2.2.1.8).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value **META_CREATEBRUSHINDIRECT**.

LogBrush (8 bytes): LogBrush Object data that defines the brush to create. The **BrushStyle** field specified in the LogBrush Object SHOULD be BS_SOLID, BS_NULL, or BS_NULL; otherwise, a default Brush Object MAY be created. See the table below for details.

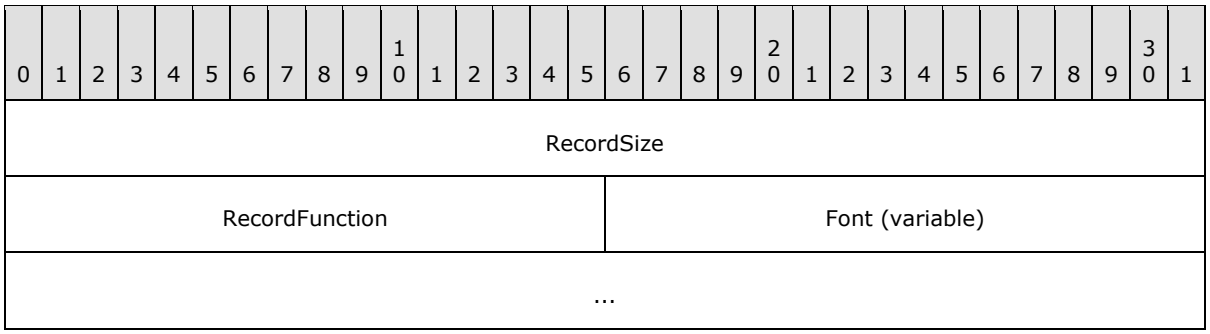
The following table shows the types of Brush Objects created by the META_CREATEBRUSHINDIRECT record, according to the **BrushStyle Enumeration (section 2.1.4)** value in the LogBrush Object specified by the **LogBrush** field.

BrushStyle	Brush Object Created
BS_SOLID	A solid-color Brush Object.
BS_NULL	An empty Brush Object.
BS_PATTERN	A default object, such as a solid-color black Brush Object, MAY be created.<25>
BS_DIBPATTERNPT	Same as BS_PATTERN above.
BS_HATCHED	A hatched Brush Object.

See section 2.3.4 for the specification of other object records.

2.3.4.2 META_CREATEFONTINDIRECT Record

The Windows Metafile Format (WMF) META_CREATEFONTINDIRECT record creates a Font Object (section 2.2.2.4).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

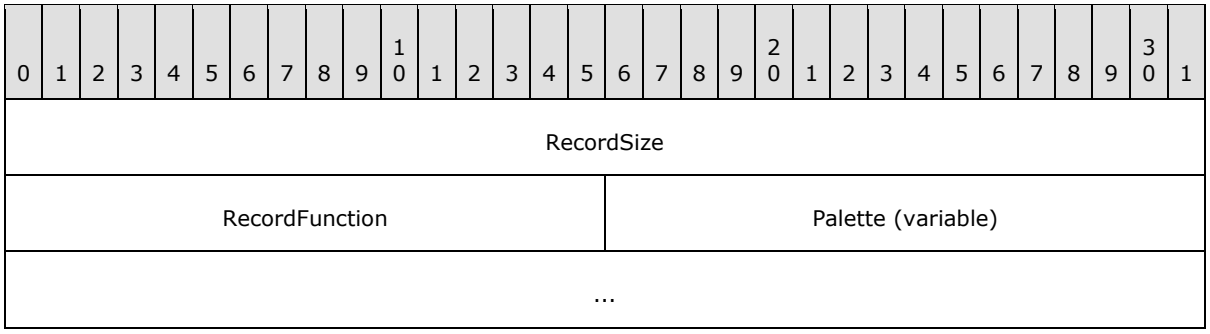
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** value **META_CREATEFONTINDIRECT**.

Font (variable): Font Object data that defines the font to create.

See section 2.3.4 for the specification of other object records.

2.3.4.3 META_CREATEPALETTE Record

The Windows Metafile Format (WMF) META_CREATEPALETTE record creates a Palette Object (section 2.2.2.7).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

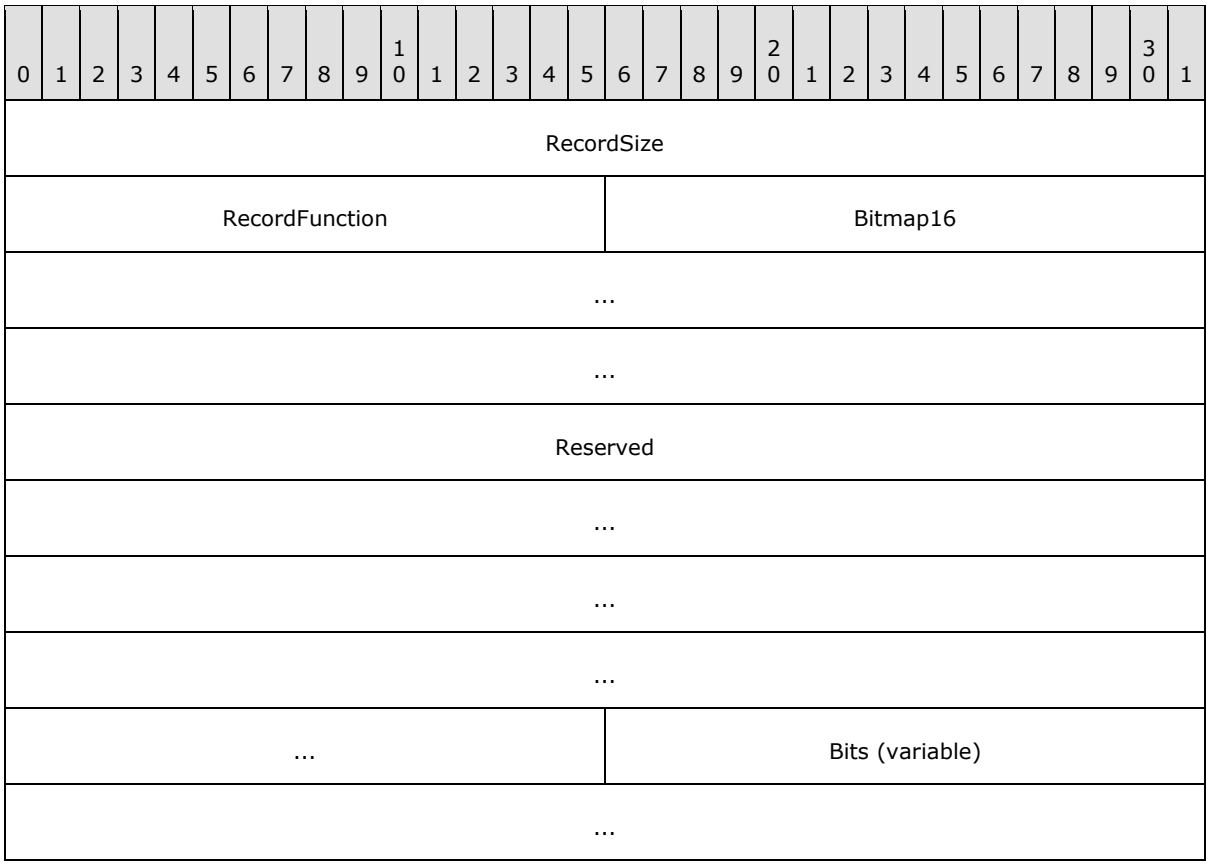
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** value **META_CREATEPALETTE**.

Palette (variable): Palette Object data that defines the palette to create. The **Start** field in the Palette Object MUST be set to 0x0300.

See section 2.3.4 for the specification of other object records.

2.3.4.4 META_CREATEPATTERNBRUSH Record

The Windows Metafile Format (WMF) META_CREATEPATTERNBRUSH record creates a pattern Brush Object (section 2.2.2.2) with a pattern specified by a Bitmap16 Object (section 2.2.2.1).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value **META_CREATEPATTERNBRUSH**.

Bitmap16 (10 bytes): The fixed-length part of a Bitmap16 Object, which is 14 bytes long. The Bitmap16 Object specifies the **Width**, **BitsPixel**, and **Height** values, which define the size of the pattern for the brush in the **Bits** field below.

Reserved (18 bytes): This field SHOULD NOT be used.

Bits (variable): An optional, variable-length array of bytes that defines the bitmap pixel data. The length of this field in bytes can be computed as follows:

```
(((Width * BitsPixel + 15) >> 4) << 1) * Height
```

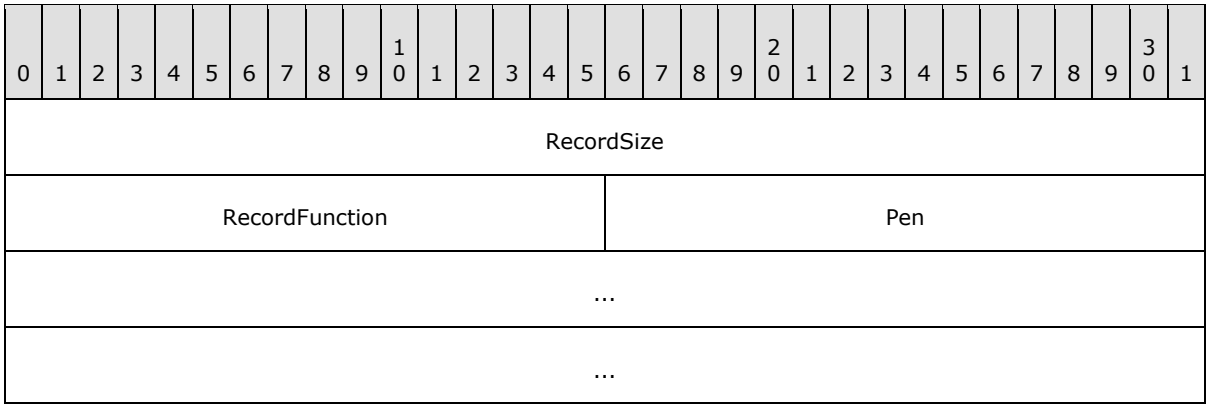
The **Width**, **BitsPixel**, and **Height** values come from the fixed-length part of the Bitmap16 Object in the **Bitmap16** field of this record.

The **BrushStyle Enumeration (section 2.1.4)** value in the Brush Object created by this record MUST be BS_PATTERN.

See section 2.3.4 for the specification of other WMF Object records.

2.3.4.5 META_CREATEPENINDIRECT Record

The Windows Metafile Format (WMF) META_CREATEPENINDIRECT record creates a Pen Object (section 2.2.1.10).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

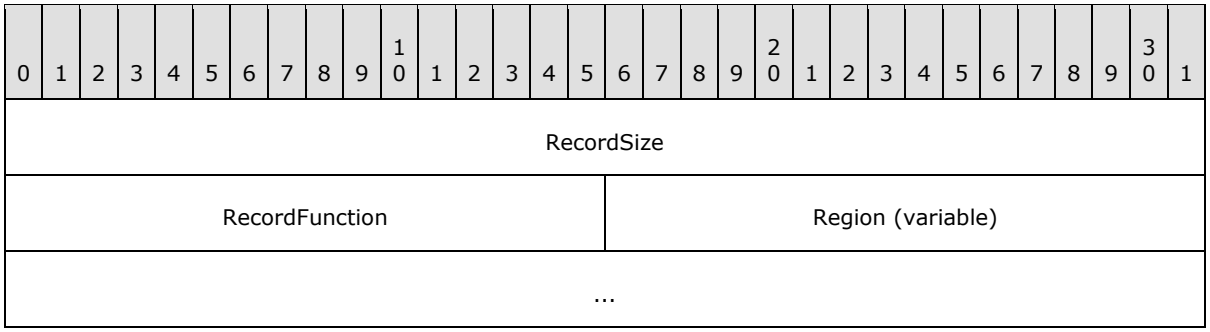
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** value **META_CREATEPENINDIRECT**.

Pen (10 bytes): Pen Object data that defines the pen to create.

See section 2.3.4 for the specification of other object records.

2.3.4.6 META_CREATEREGION Record

The Windows Metafile Format (WMF) META_CREATEREGION record creates a Region Object (section 2.2.2.9).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

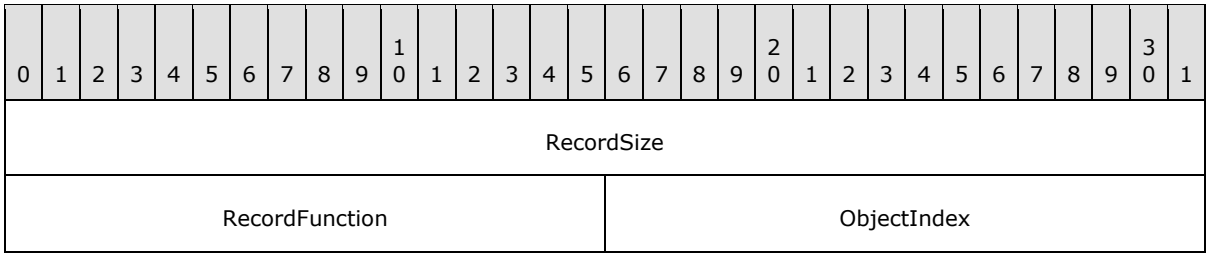
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** value **META_CREATEREGION**.

Region (variable): Region Object data that defines the region to create.

See section 2.3.4 for the specification of other object records.

2.3.4.7 META_DELETEOBJECT Record

The Windows Metafile Format META_DELETEOBJECT record deletes an object, including Bitmap16, Brush, DeviceIndependentBitmap, Font, Palette, Pen and Region. After the object is deleted, its index in the WMF Object Table is no longer valid but is available to be reused.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_DELETEOBJECT.

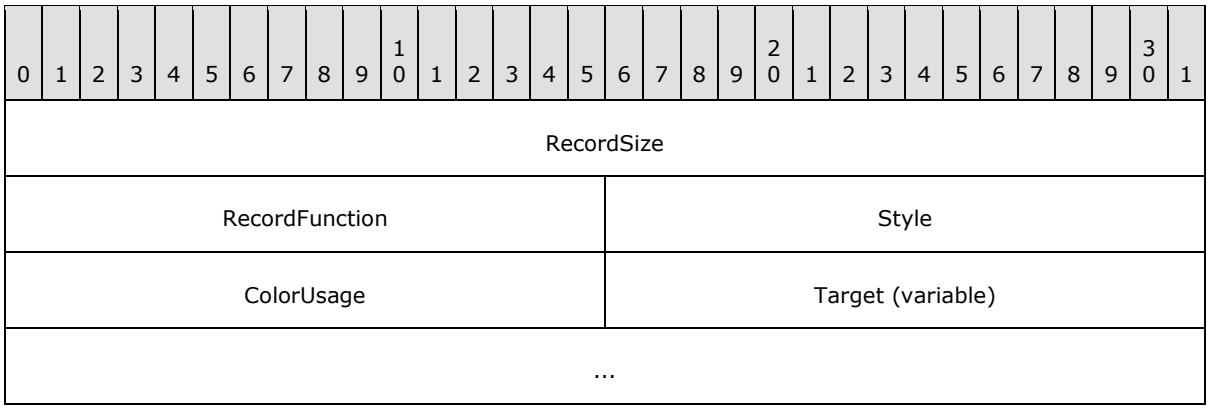
ObjectIndex (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the object to be deleted.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.4 for the specification of other Object records.

2.3.4.8 META_DIBCREATEPATTERNBRUSH Record

The Windows Metafile Format (WMF) META_DIBCREATEPATTERNBRUSH record creates a Brush Object (section 2.2.2.2) with a pattern specified by a DeviceIndependentBitmap (DIB) Object (section 2.2.2.3).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value **META_DIBCREATEPATTERNBRUSH**.

Style (2 bytes): A 16-bit unsigned integer that defines the brush style. The legal values for this field are defined as follows: if the value is not BS_PATTERN, BS_DIBPATTERNPT MUST be assumed. These values are specified in the **BrushStyle Enumeration (section 2.1.4)**.

ColorUsage (2 bytes): A 16-bit unsigned integer that defines whether the **Colors** field of a DIB Object contains explicit red, green, and blue (RGB) values, or indexes into a palette.

If the **Style** field specifies BS_PATTERN, a **ColorUsage** value of DIB_RGB_COLORS MUST be used regardless of the contents of this field.

If the **Style** field specified anything but BS_PATTERN, this field MUST be one of the values in the **ColorUsage Enumeration (section 2.1.7)**.

Target (variable): Variable-bit DIB Object data that defines the pattern to use in the brush.

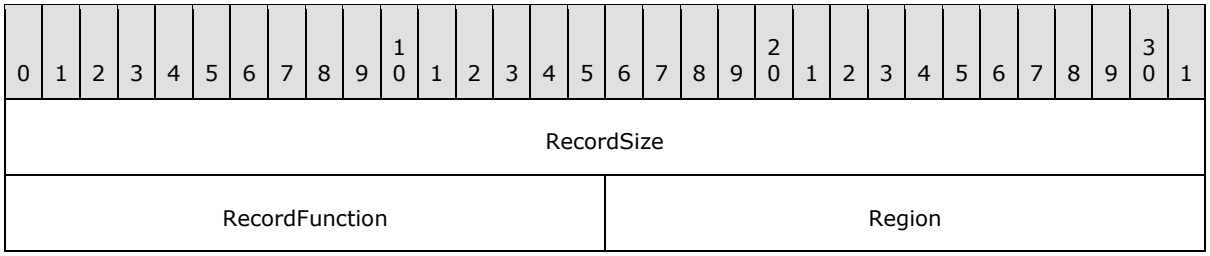
The following table shows the types of Brush Objects created by the META_DIBCREATEPATTERNBRUSH record, according to **BrushStyle Enumeration** values.

BrushStyle	ColorUsage	Brush Object Created
BS_SOLID	Same as BS_DIBPATTERNPT below.	Same as BS_DIBPATTERNPT below.
BS_NULL	Same as BS_DIBPATTERNPT below.	Same as BS_DIBPATTERNPT below.
BS_PATTERN	A **ColorUsage Enumeration** value, which SHOULD define how to interpret the logical color values in the brush pattern.	A BS_PATTERN Brush Object that SH0ULD contain a pattern defined by a Bitmap16 Object.
BS_DIBPATTERNPT	A **ColorUsage Enumeration** value, which SHOULD define how to interpret the logical color values in the brush pattern.	A BS_DIBPATTERNPT Brush Object that SHOULD contain a pattern defined by a DIB Object.
BS_HATCHED	Same as BS_DIBPATTERNPT above.	Same as BS_DIBPATTERNPT above.

See section 2.3.4 for the specification of other object records.

2.3.4.9 META_SELECTCLIPREGION Record

The Windows Metafile Format META_SELECTCLIPREGION record specifies a Region Object (section 2.2.2.9) to be the current clipping region.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SELECTCLIPREGION.

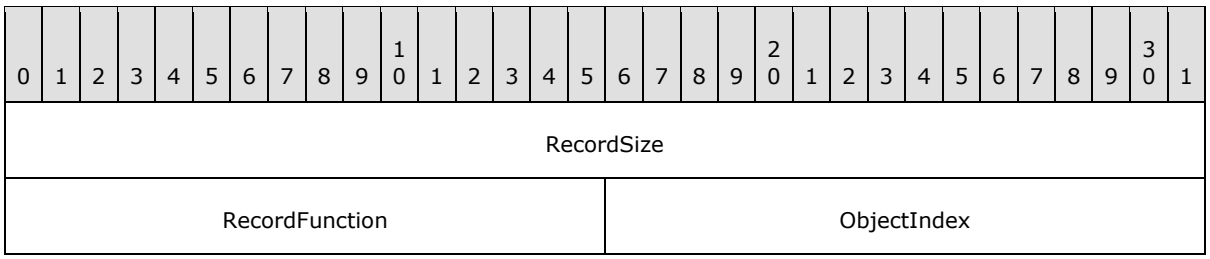
Region (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the region to be inverted.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.4 for the specification of other Object records.

2.3.4.10 META_SELECTOBJECT Record

The Windows Metafile Format META_SELECTOBJECT record specifies a graphics object for the playback device context. The new object replaces the previous object of the same type.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SELECTOBJECT.

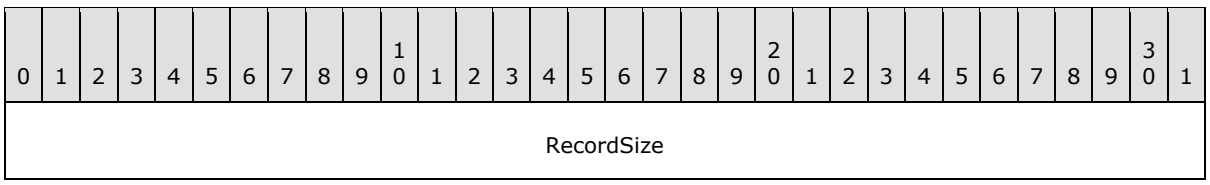
ObjectIndex (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the object to be selected.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.4 for the specification of other Object records.

2.3.4.11 META_SELECTPALETTE Record

The Windows Metafile Format META_SELECTPALETTE record defines the current logical palette with a specified Palette Object.



RecordFunction	Palette

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SELECTPALETTE.

Palette (2 bytes): A 16-bit unsigned integer used to index into the WMF Object Table to get the Palette Object to be selected.

The WMF Object Table refers to an indexed table of WMF Objects that are defined in the metafile. See section 3.1.4.1 for more information.

See section 2.3.4 for the specification of other Object records.

2.3.5 State Record Types

This section defines the Windows Metafile Format (WMF) State record types, which define and manage the properties of the graphics state.

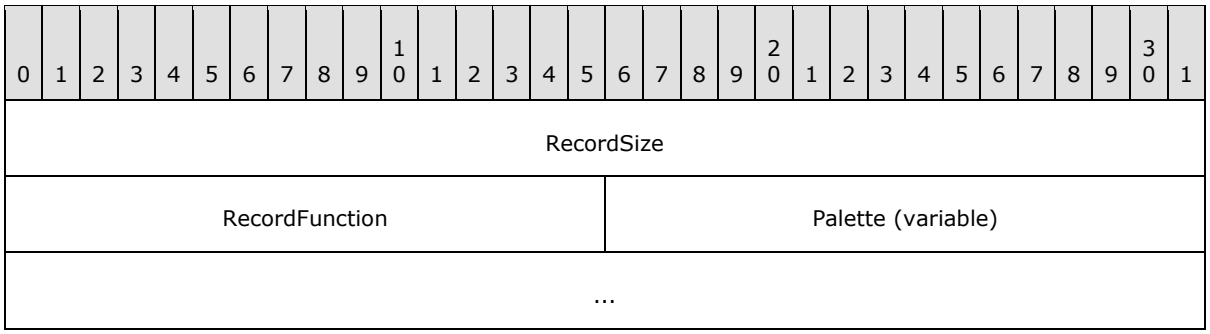
The following are the State record types:

Name	Section	Description
META_ANIMATEPALETTE	2.3.5.1	Defines the logical palette that is currently selected into the device context with the specified Palette Object.
META_EXCLUDECLIPRECT	2.3.5.2	Defines a clipping region that consists of the existing clipping region minus a specified rectangle.
META_INTERSECTCLIPRECT	2.3.5.3	Defines a clipping region from the intersection of the current clipping region and a specified rectangle.
META_MOVETO	2.3.5.4	Defines the current drawing position to be the specified point.
META_OFFSETCLIPRGN	2.3.5.5	Defines the current clipping region with the specified offsets.
META_OFFSETVIEWPORTORG	2.3.5.6	Defines the viewport origin using the specified horizontal and vertical offsets.
META_OFFSETWINDOWORG	2.3.5.7	Defines the window origin using the specified horizontal and vertical offsets.
META_REALIZEPALETTE	2.3.5.8	Maps entries from the current logical palette to the system palette.
META_RESIZEPALETTE	2.3.5.9	Redefines the size of a logical palette.
META_RESTOREDC	2.3.5.10	Defines the current device context from a previously-saved device context.
META_SAVEDC	2.3.5.11	Saves the current device context for later retrieval.
META_SCALEVIEWPORTEXT	2.3.5.12	Defines the current viewport using the ratios formed by

Name	Section	Description
		specified multiplicands and divisors.
META_SCALEWINDOWEXT	2.3.5.13	Defines the current window using the ratios formed by specified multiplicands and divisors.
META_SETBKCOLOR	2.3.5.14	Sets the current background color to the specified color value.
META_SETBKMODE	2.3.5.15	Defines the current background mix mode.
META_SETLAYOUT	2.3.5.16	Defines layout orientation options.
META_SETMAPMODE	2.3.5.17	Defines the current mapping mode.
META_SETMAPPERFLAGS	2.3.5.18	Defines the algorithm that the font mapper uses when it maps logical fonts to physical fonts.
META_SETPALENTRIES	2.3.5.19	Defines RGB color values and flags in a range of entries in a logical palette.
META_SETPOLYFILLMODE	2.3.5.20	Defines the polygon fill mode for functions that fill polygons.
META_SETRELABS	2.3.5.21	Reserved and MUST NOT be used.
META_SETROP2	2.3.5.22	Defines the current foreground binary raster operation mixing mode.
META_SETSTRETCHBLTMODE	2.3.5.23	Defines the current bitmap stretching mode.
META_SETTEXTALIGN	2.3.5.24	Defines the current text-alignment values.
META_SETTEXTCHAREXTRA	2.3.5.25	Defines intercharacter spacing for text justification.
META_SETTEXTCOLOR	2.3.5.26	Defines the current text foreground color.
META_SETTEXTJUSTIFICATION	2.3.5.27	Defines the amount of space to add to break characters in a string of justified text.
META_SETVIEWPORTEXT	2.3.5.28	Defines the horizontal and vertical extents of the current viewport.
META_SETVIEWPORTORG	2.3.5.29	Defines the current viewport origin (0,0).
META_SETWINDOWEXT	2.3.5.30	Defines the horizontal and vertical extents of the current output window.
META_SETWINDOWORG	2.3.5.31	Defines the current window origin (0,0).

2.3.5.1 META_ANIMATEPALETTE Record

The Windows Metafile Format META_ANIMATEPALETTE record redefines the logical palette that is currently selected into the device context with the specified Palette Object.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

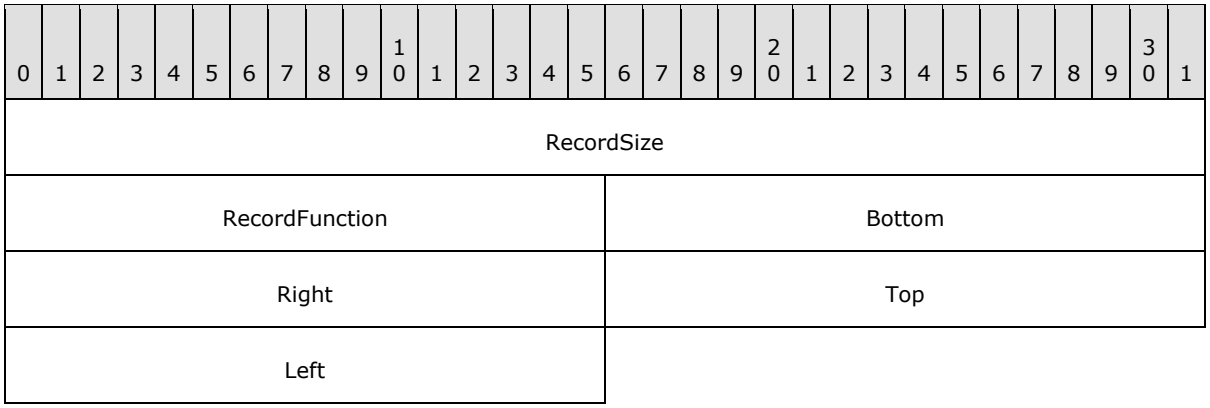
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ANIMATEPALETTE.

Palette (variable): A variable-sized Palette Object that specifies a logical palette.

See section 2.3.5 for the specification of other State record types.

2.3.5.2 META_EXCLUDECLIPRECT Record

The Windows Metafile Format META_EXCLUDECLIPRECT record defines a clipping region that consists of the existing clipping region minus the specified rectangle.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_EXCLUDECLIPRECT.

Bottom (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the rectangle.

Right (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the rectangle.

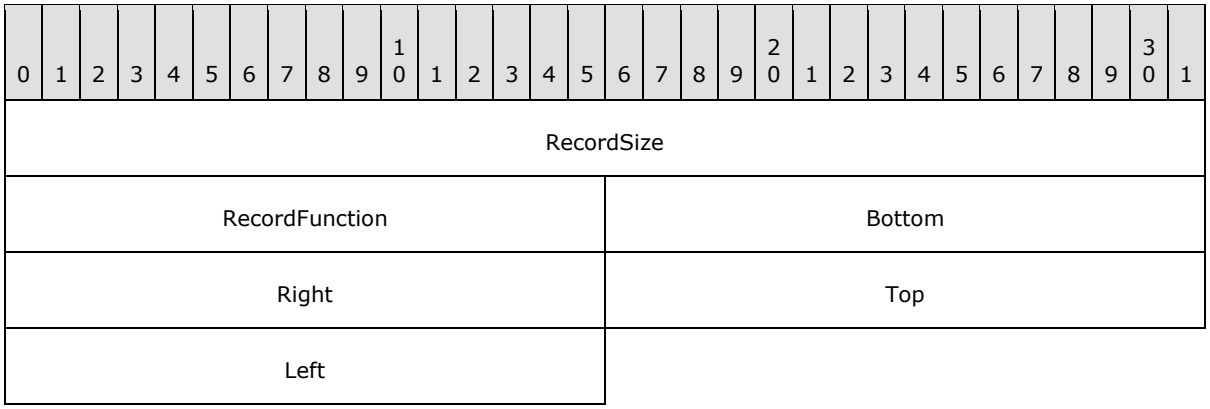
Top (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the rectangle.

Left (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle.

See section 2.3.5 for the specification of other State record types.

2.3.5.3 META_INTERSECTCLIPRECT Record

The Windows Metafile Format META_INTERSECTCLIPRECT record defines a clipping region from the intersection of the current clipping region and the specified rectangle.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_INTERSECTCLIPRECT.

Bottom (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the lower-right corner of the rectangle.

Right (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the lower-right corner of the rectangle.

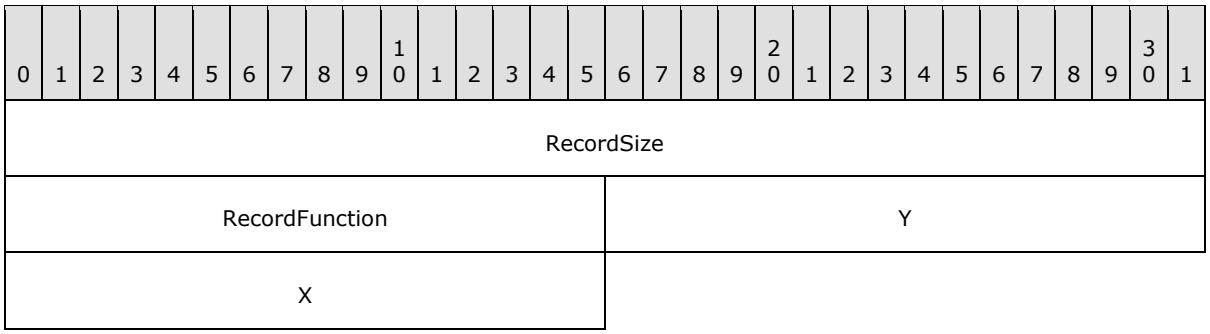
Top (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units, of the upper-left corner of the rectangle.

Left (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units, of the upper-left corner of the rectangle.

See section 2.3.5 for the specification of other State record types.

2.3.5.4 META_MOVETO Record

The Windows Metafile Format META_MOVETO record redefines the current position to the specified point.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_MOVETO.

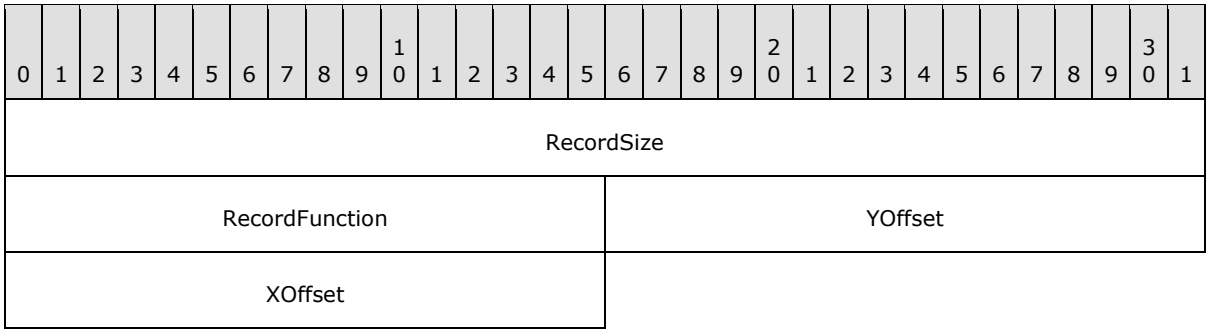
Y (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units.

X (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units.

See section 2.3.5 for the specification of other State record types.

2.3.5.5 META_OFFSETCLIPRGN Record

The Windows Metafile Format META_OFFSETCLIPRGN record redefines the clipping region of a device context by the specified offsets.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_OFFSETCLIPRGN.

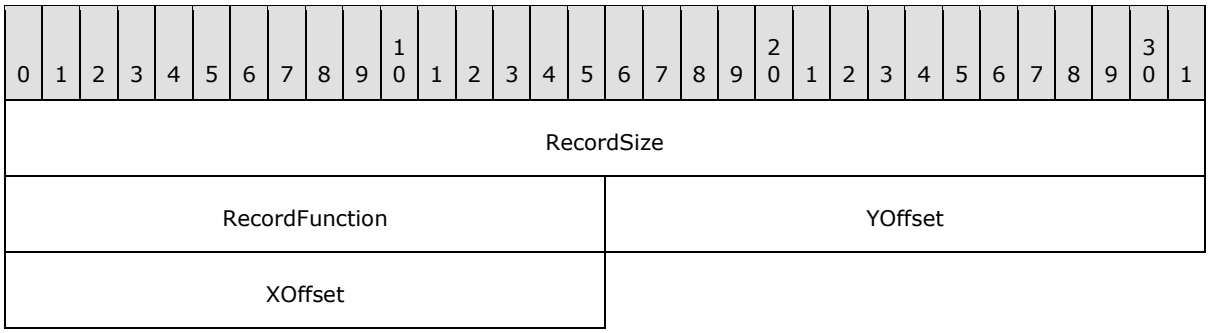
YOffset (2 bytes): A 16-bit signed integer that defines the number of logical units to move up or down.

XOffset (2 bytes): A 16-bit signed integer that defines the number of logical units to move left or right.

See section 2.3.5 for the specification of other State record types.

2.3.5.6 META_OFFSETVIEWPORTORG Record

The Windows Metafile Format META_OFFSETVIEWPORTORG record redefines the viewport origin for a device context using the specified horizontal and vertical offsets.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_OFFSETVIEWPORTORG.

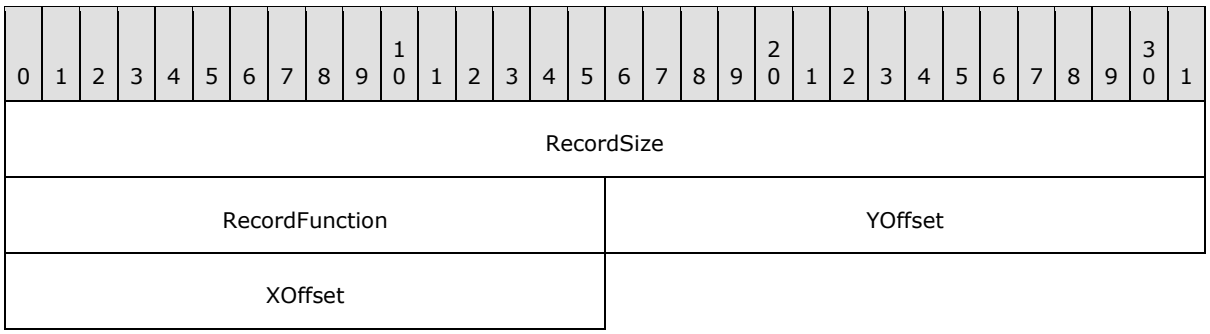
YOffset (2 bytes): A 16-bit signed integer that defines the vertical offset, in device units.

XOffset (2 bytes): A 16-bit signed integer that defines the horizontal offset, in device units.

See section 2.3.5 for the specification of other State record types.

2.3.5.7 META_OFFSETWINDOWORG Record

The Windows Metafile Format META_OFFSETWINDOWORG record redefines the window origin for a device context using the specified horizontal and vertical offsets.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_OFFSETWINDOWORG.

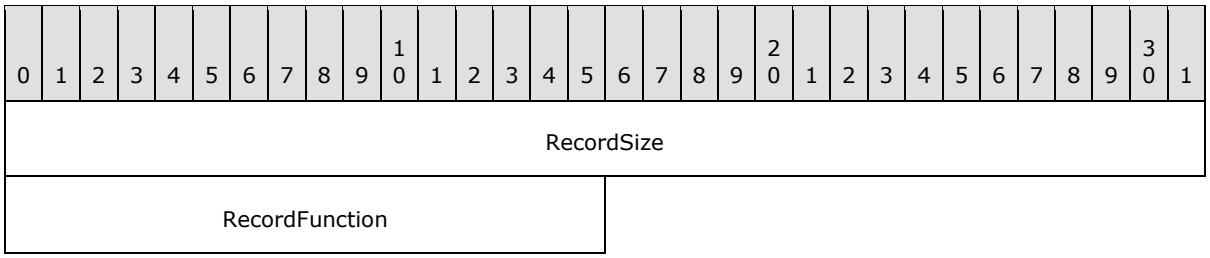
YOffset (2 bytes): A 16-bit signed integer that defines the vertical offset, in device units.

XOffset (2 bytes): A 16-bit signed integer that defines the horizontal offset, in device units.

See section 2.3.5 for the specification of other State record types.

2.3.5.8 META_REALIZEPALETTE Record

The Windows Metafile Format META_REALIZEPALETTE record maps entries from the current logical palette to the system palette.



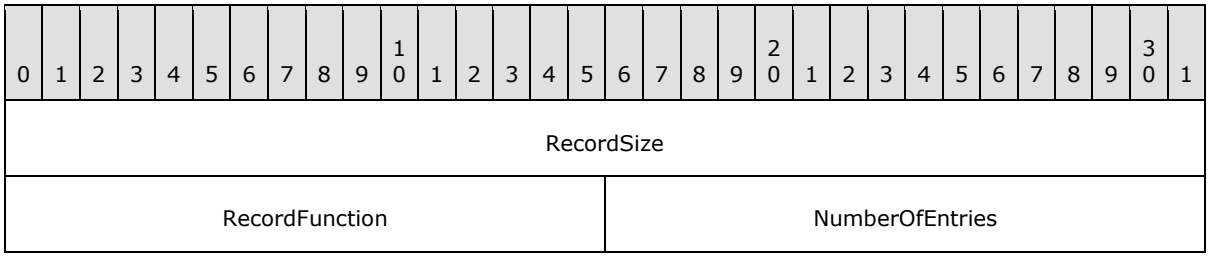
RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_REALIZEPALETTE.

See section 2.3.5 for the specification of other State record types.

2.3.5.9 META_RESIZEPALETTE Record

The Windows Metafile Format META_RESIZEPALETTE record redefines the size of a logical palette based on the specified value.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

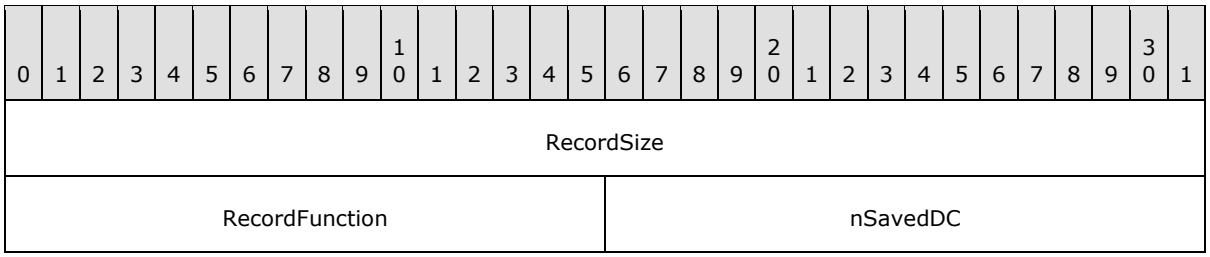
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_RESIZEPALETTE.

NumberOfEntries (2 bytes): A 16-bit unsigned integer that defines the number of entries in the logical palette.

See section 2.3.5 for the specification of other State record types.

2.3.5.10 META_RESTOREDC Record

The Windows Metafile Format META_RESTOREDC record defines a record that the current device context should be set from a previously saved device context.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

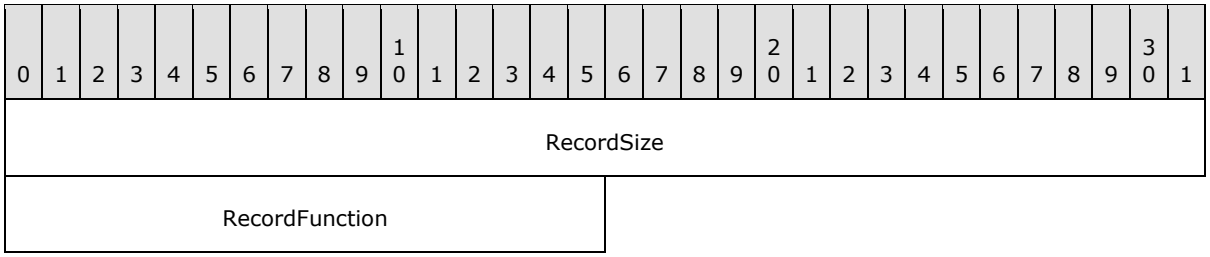
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_RESTOREDC.

nSavedDC (2 bytes): A 16-bit signed integer that defines the saved state to be restored. If this member is positive, **nSavedDC** represents a specific instance of the state to be restored. If this member is negative, **nSavedDC** represents an instance relative to the current state.

See section 2.3.5 for the specification of other State record types.

2.3.5.11 META_SAVEDC Record

The Windows Metafile Format META_SAVEDC record defines a record that the current device context should be saved for later retrieval.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SAVEDC.

See section 2.3.5 for the specification of other State record types.

2.3.5.12 META_SCALEVIEWPORTEXT Record

The Windows Metafile Format META_SCALEVIEWPORTEXT record defines the viewport for a device context by using the ratios formed by the specified multiplicands and divisors.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																yDenom															
yNum																xDenom															
xNum																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SCALEVIEWPORTEXT.

yDenom (2 bytes): A 16-bit signed integer that defines the amount by which to divide the result of multiplying the current y-extent by the value of the **yNum** member.

yNum (2 bytes): A 16-bit signed integer that defines the amount by which to multiply the current y-extent.

xDenom (2 bytes): {annotation author="a-jmicka" time="8/20/2007 1:21:22 PM"}L-S: See suggestion above and use here as well. A 16-bit signed integer that defines the amount by which to divide the result of multiplying the current x-extent by the value of the **xNum** member.

xNum (2 bytes): A 16-bit signed integer that defines the amount by which to multiply the current x-extent.

See section 2.3.5 for the specification of other State record types.

2.3.5.13 META_SCALEWINDOWEXT Record

The Windows Metafile Format META_SCALEWINDOWEXT record redefines the window for a device context by using the ratios formed by the specified multiplicands and divisors.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																yDenom															
yNum																xDenom															
xNum																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SCALEWINDOWEXT.

yDenom (2 bytes): A 16-bit signed integer that defines the amount by which to divide the result of multiplying the current y-extent by the value of the **yNum** member.

yNum (2 bytes): A 16-bit signed integer that defines the amount by which to multiply the current y-extent.

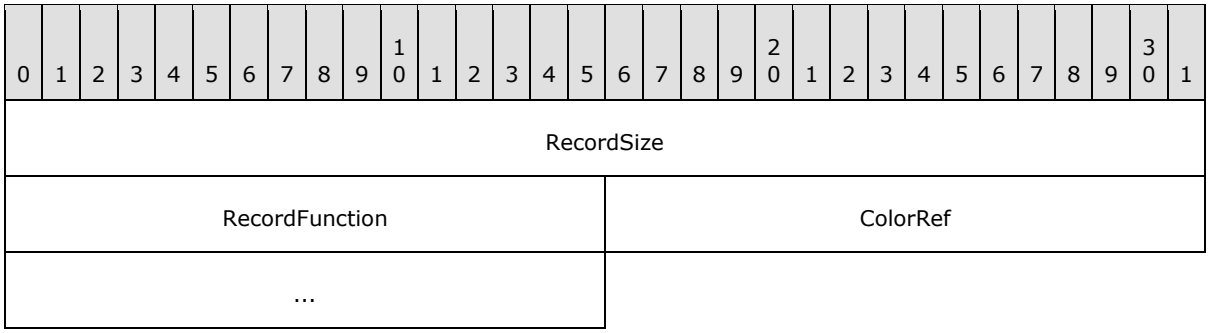
xDenom (2 bytes): A 16-bit signed integer that defines the amount by which to divide the result of multiplying the current x-extent by the value of the **xNum** member.

xNum (2 bytes): A 16-bit signed integer that defines the amount by which to multiply the current x-extent.

See section 2.3.5 for the specification of other State record types.

2.3.5.14 META_SETBKCOLOR Record

The Windows Metafile Format META_SETBKCOLOR record defines current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

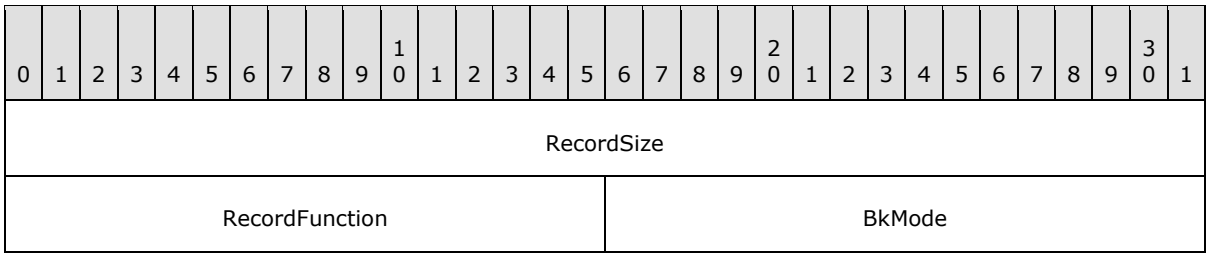
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETBKCOLOR.

ColorRef (4 bytes): A 32-bit ColorRef Object that defines the color value.

See section 2.3.5 for the specification of other State record types.

2.3.5.15 META_SETBKMODE Record

The Windows Metafile Format META_SETBKMODE record defines the background mix mode of the current graphics context. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

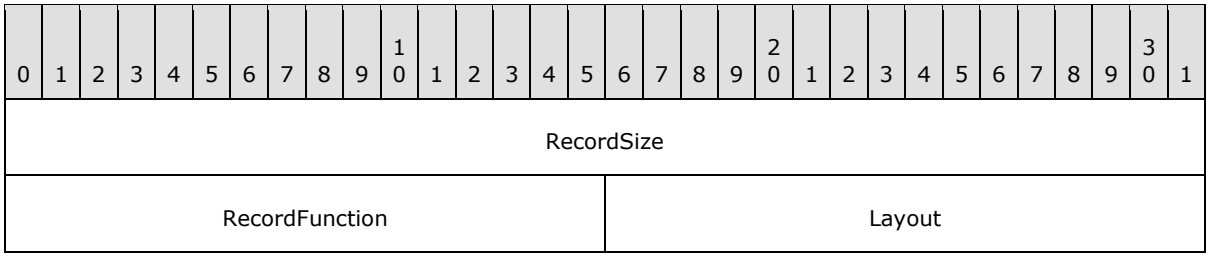
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETBKMODE.

BkMode (2 bytes): A 16-bit unsigned integer that defines background mix mode. This MUST be one of the values in the **MixMode Enumeration** table.

See section 2.3.5 for the specification of other State record types.

2.3.5.16 META_SETLAYOUT Record

The Windows Metafile Format META_SETLAYOUT record defines layout orientation options.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

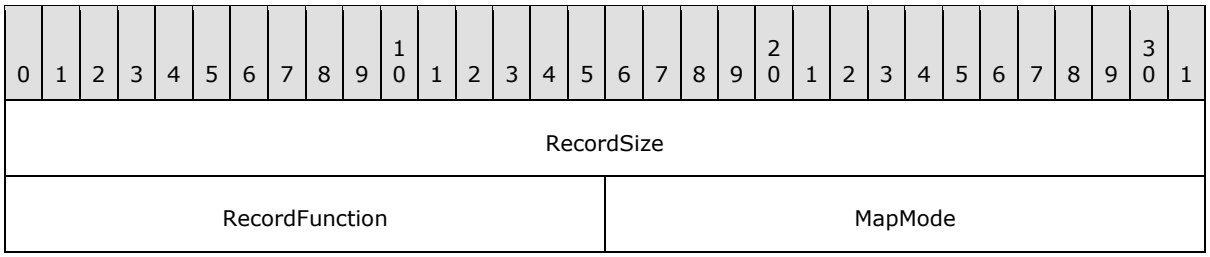
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETLAYOUT.

Layout (2 bytes): A 16-bit unsigned integer that defines the layout orientation options. The value MUST be in the **Layout Enumeration** table.

See section 2.3.5 for the specification of other State record types.

2.3.5.17 META_SETMAPMODE Record

The Windows Metafile Format META_SETMAPMODE record defines the mapping mode of the specified device context. The mapping mode defines the unit of measure used to transform page-space units into device-space units, and also defines the orientation of the device's x and y axes.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

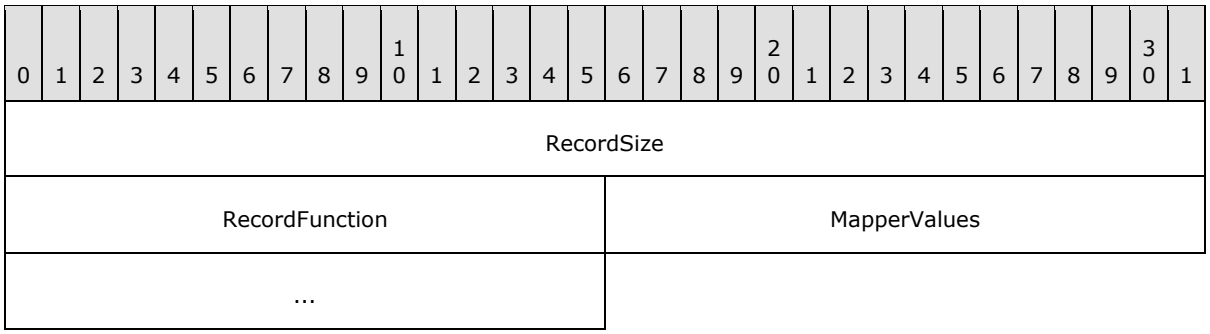
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETMAPMODE.

MapMode (2 bytes): A 16-bit unsigned integer that defines background mix mode. This MUST be one of the values enumerated in the **MapMode Enumeration** table.

See section 2.3.5 for the specification of other State record types.

2.3.5.18 META_SETMAPPERFLAGS Record

The Windows Metafile Format META_SETMAPPERFLAGS record redefines the algorithm that the font mapper uses when it maps logical fonts to physical fonts.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

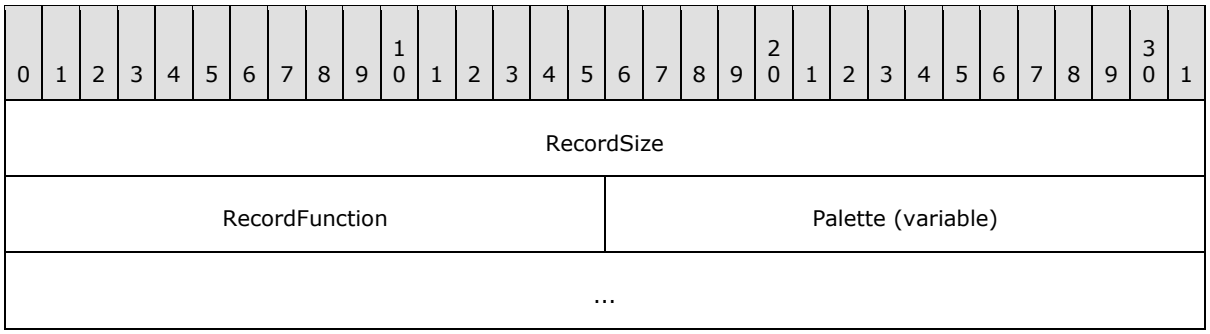
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETMAPPERFLAGS.

MapperValues (4 bytes): A 32-bit unsigned integer that defines whether the font mapper should attempt to match a font's aspect ratio to the current device's aspect ratio. If bit 0 is set, the mapper selects only matching fonts.

See section 2.3.5 for the specification of other State record types.

2.3.5.19 META_SETPALENTRIES Record

The Windows Metafile Format META_SETPALENTRIES record defines RGB (red, green, blue) color values and flags in a range of entries in a logical palette.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

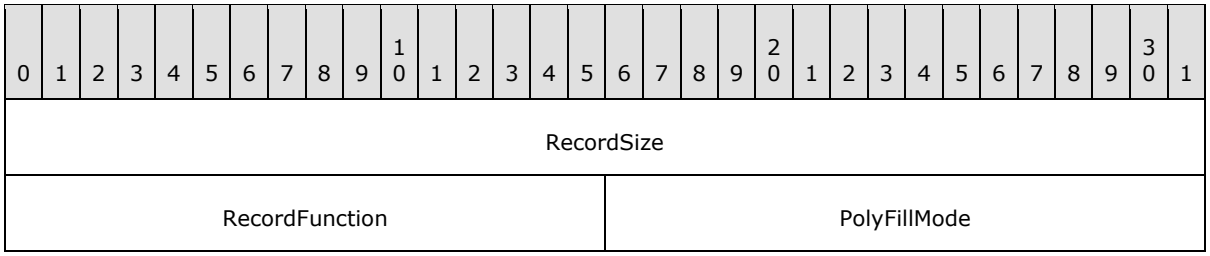
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETPALENTRIES.

Palette (variable): A variable-length Palette Object that defines the palette information.

See section 2.3.5 for the specification of other State record types.

2.3.5.20 META_SETPOLYFILLMODE Record

The Windows Metafile Format META_SETPOLYFILLMODE record defines the polygon fill mode for functions that fill polygons.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

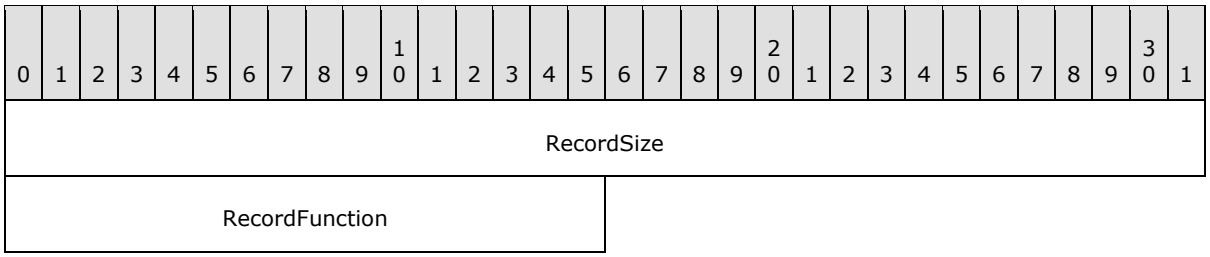
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETPOLYFILLMODE.

PolyFillMode (2 bytes): A 16-bit unsigned integer that defines background mix mode. This MUST be one of the values in the **PolyFileMode** enumeration table.

See section 2.3.5 for the specification of other State record types.

2.3.5.21 META_SETRELABS Record

The Windows Metafile Format META_SETRELABS record is reserved and not supported.



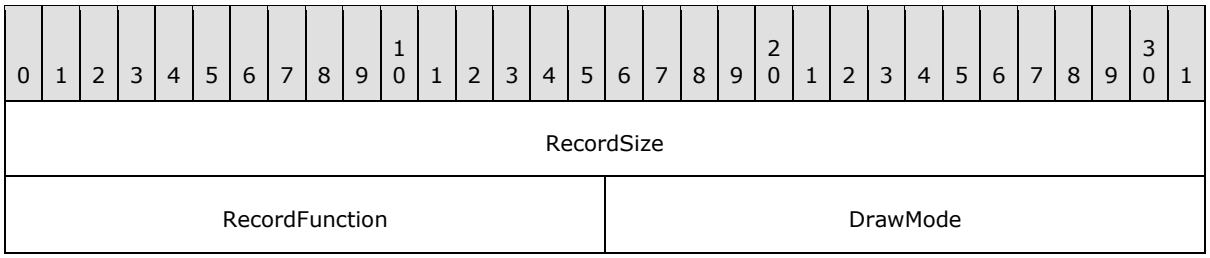
RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETRELABS.

See section 2.3.5 for the specification of other State record types.

2.3.5.22 META_SETROP2 Record

The Windows Metafile Format META_SETROP2 record defines the current foreground binary raster operation mixing mode. Metafile processing uses the foreground mix mode to combine pens and interiors of filled objects with the colors already on the screen. The foreground mix mode defines how the colors from the brush or pen and the colors in the existing image are to be combined.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

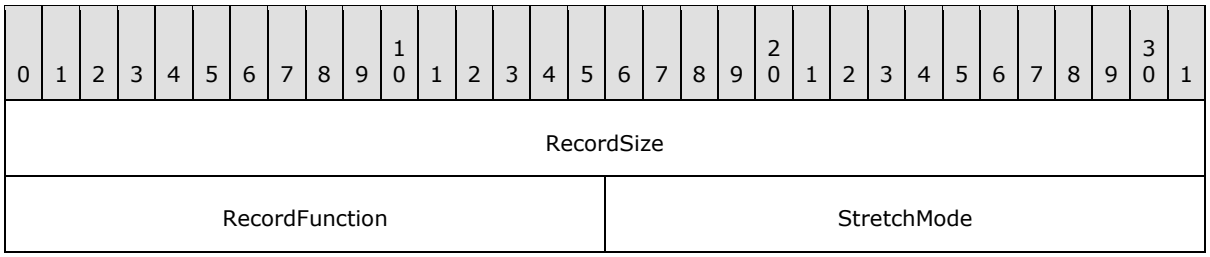
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETROP2.

DrawMode (2 bytes): A 16-bit unsigned integer that defines background mix mode. This MUST be one of the values in the **Binary Raster Operation** enumeration table.

See section 2.3.5 for the specification of other State record types.

2.3.5.23 META_SETSTRETCHBLTMODE Record

The Windows Metafile Format META_SETSTRETCHBLTMODE record defines the bitmap stretching mode in the specified device context.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

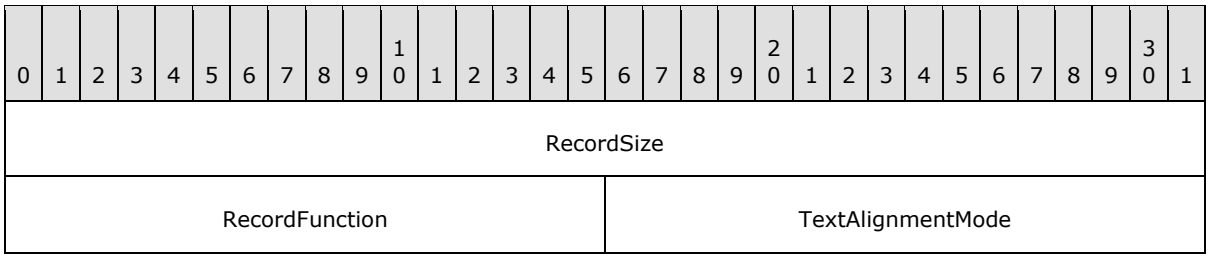
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value **META_SETSTRETCHBLTMODE**.

StretchMode (2 bytes): A 16-bit unsigned integer that defines bitmap stretching mode. This MUST be one of the values in the **StretchMode Enumeration** table.

See section 2.3.5 for the specification of other State record types.

2.3.5.24 META_SETTEXTALIGN Record

The Windows Metafile Format META_SETTEXTALIGN record defines the text-alignment values for the specified device context.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

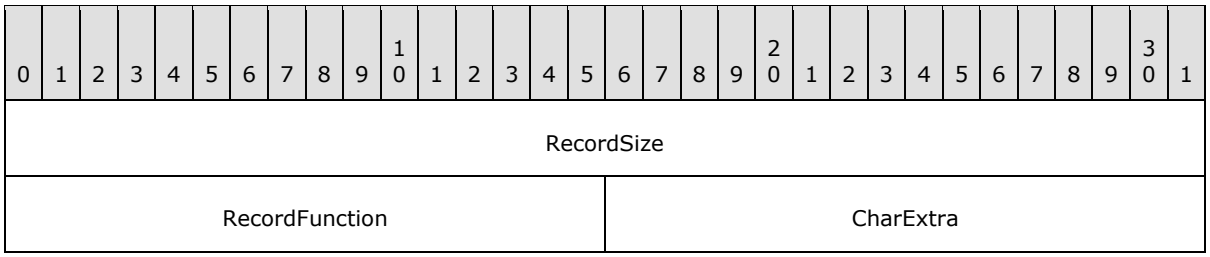
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETTEXTALIGN.

TextAlignmentMode (2 bytes): A 16-bit unsigned integer that defines text alignment. The value MUST be a combination of one or more values in the **TextAlignment Mode** enumeration table.

See section 2.3.5 for the specification of other State record types.

2.3.5.25 META_SETTEXTCHAREXTRA Record

The Windows Metafile Format (WMF) META_SETTEXTCHAREXTRA record defines intercharacter spacing. Intercharacter spacing is added to each character, including break characters, when the system writes a line of text.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

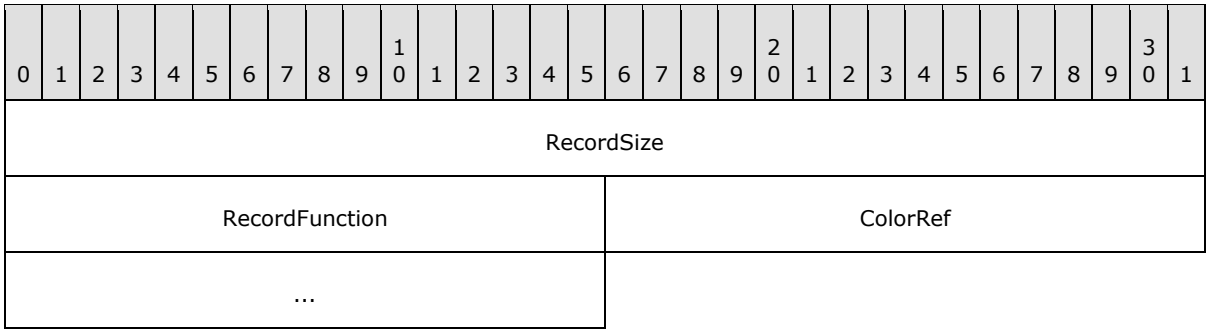
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value **META_SETTEXTCHAREXTRA**.

CharExtra (2 bytes): A 16-bit unsigned integer that defines the amount of extra space, in logical units, to be added to each character. If the current mapping mode is not MM_TEXT, this value is transformed and rounded to the nearest pixel. For details about setting the mapping mode, see META_SETMAPMODE (section 2.3.5.17).

See section 2.3.5 for the specification of other state record types.

2.3.5.26 META_SETTEXTCOLOR Record

The Windows Metafile Format META_SETTEXTCOLOR record defines the text color for the specified device context to the specified color.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

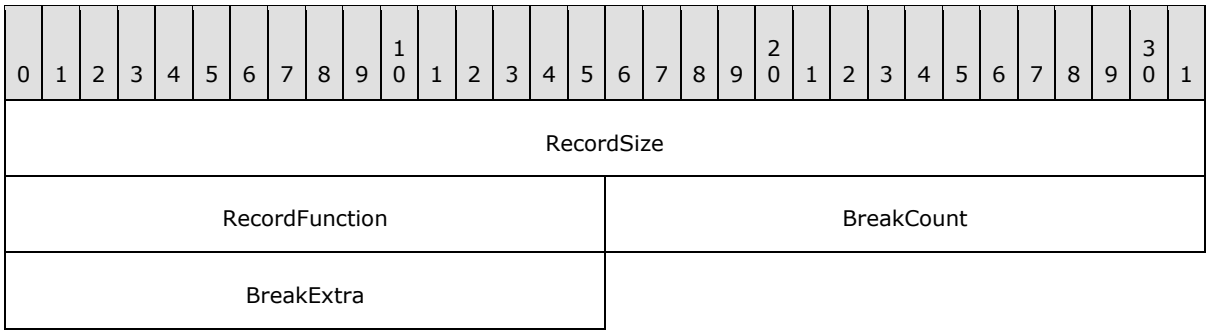
RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETTEXTCOLOR.

ColorRef (4 bytes): A 32-bit ColorRef Object that defines the color value.

See section 2.3.5 for the specification of other State record types.

2.3.5.27 META_SETTEXTJUSTIFICATION Record

The Windows Metafile Format (WMF) META_SETTEXTJUSTIFICATION record defines the amount of space to add to the break characters in a string of text.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** value **META_SETTEXTJUSTIFICATION**.

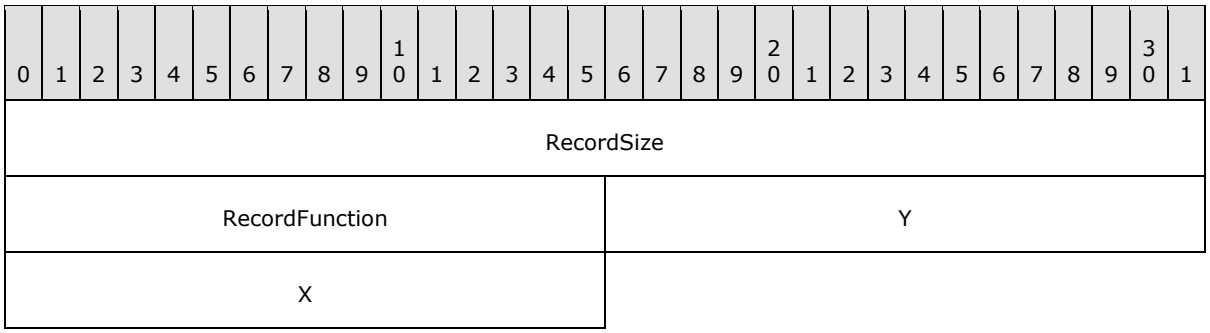
BreakCount (2 bytes): A 16-bit unsigned integer that specifies the number of space characters in the line.

BreakExtra (2 bytes): A 16-bit unsigned integer that specifies the total extra space, in logical units, to be added to the line of text. If the current mapping mode is not MM_TEXT, the value identified by the **BreakExtra** member is transformed and rounded to the nearest pixel. For details about setting the mapping mode, see META_SETMAPMODE (section 2.3.5.17).

See section 2.3.5 for the specification of other State record types.

2.3.5.28 META_SETVIEWPORTEXT Record

The Windows Metafile Format META_SETVIEWPORTEXT record defines the horizontal and vertical extents of the viewport for a device context by using the specified values.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETVIEWPORTEXT.

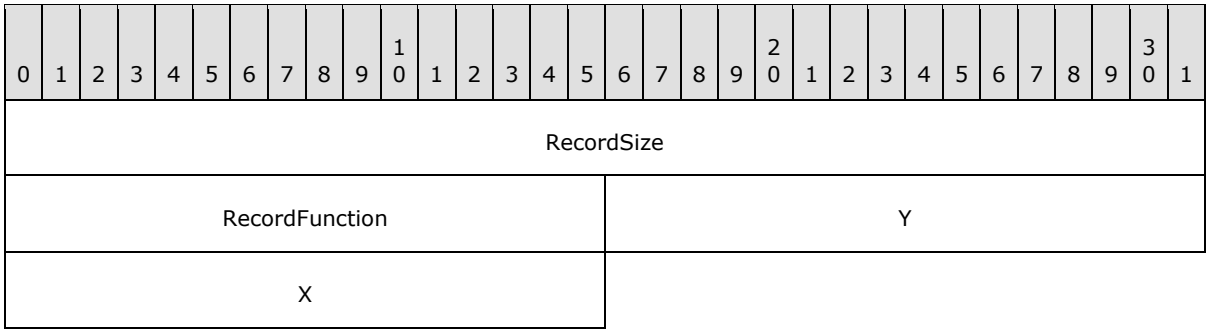
Y (2 bytes): A 16-bit signed integer that defines the vertical extent of the viewport in device units.

X (2 bytes): A 16-bit signed integer that defines the horizontal extent of the viewport in device units.

See section 2.3.5 for the specification of other State record types.

2.3.5.29 META_SETVIEWPORTORG Record

The Windows Metafile Format META_SETVIEWPORTORG record defines which device point maps to the viewport origin (0,0).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETVIEWPORTORG.

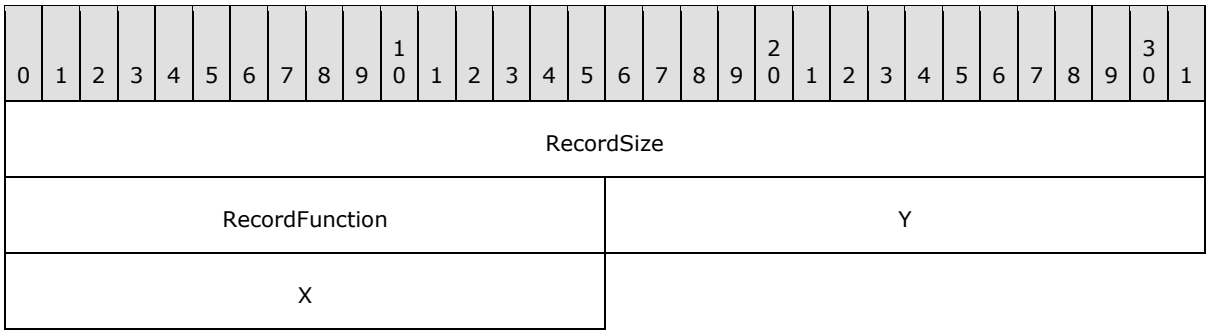
Y (2 bytes): A 16-bit signed integer that defines y-coordinate, in logical units.

X (2 bytes): A 16-bit signed integer that defines x-coordinate, in logical units.

See section 2.3.5 for the specification of other State record types.

2.3.5.30 META_SETWINDOWEXT Record

The Windows Metafile Format META_SETWINDOWEXT record defines the horizontal and vertical extents of the window for a device context by using the specified values.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETWINDOWEXT.

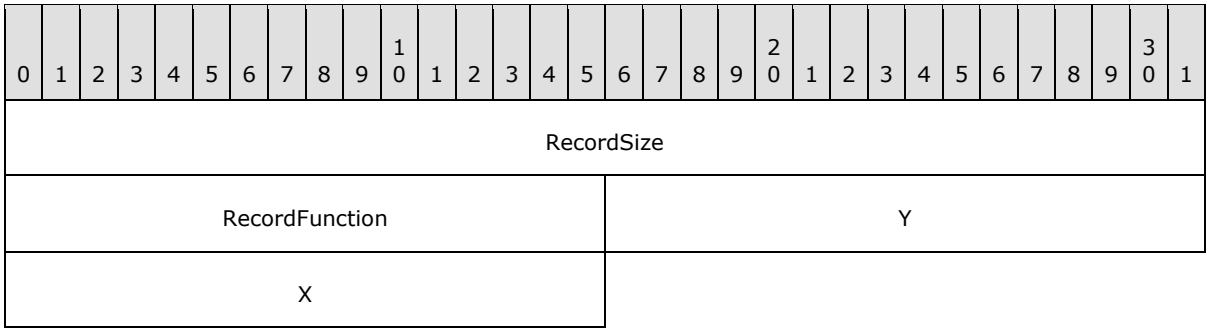
Y (2 bytes): A 16-bit signed integer that defines the vertical extent of the window in logical units.

X (2 bytes): A 16-bit signed integer that defines the horizontal extent of the window in logical units.

See section 2.3.5 for the specification of other State record types.

2.3.5.31 META_SETWINDOWORG Record

The Windows Metafile Format META_SETWINDOWORG record defines which window point maps to the window origin (0,0).



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_SETWINDOWORG.

Y (2 bytes): A 16-bit signed integer that defines the y-coordinate, in logical units.

X (2 bytes): A 16-bit signed integer that defines the x-coordinate, in logical units.

See section 2.3.5 for the specification of other State record types.

2.3.6 Escape Record Types

This section defines the Windows Metafile Format (WMF) Escape Record Types, which specify extensions to metafile functionality. The generic format of all Escape records is specified in the META_ESCAPE (section 2.3.6.1) record.

The following are Escape Record Types:

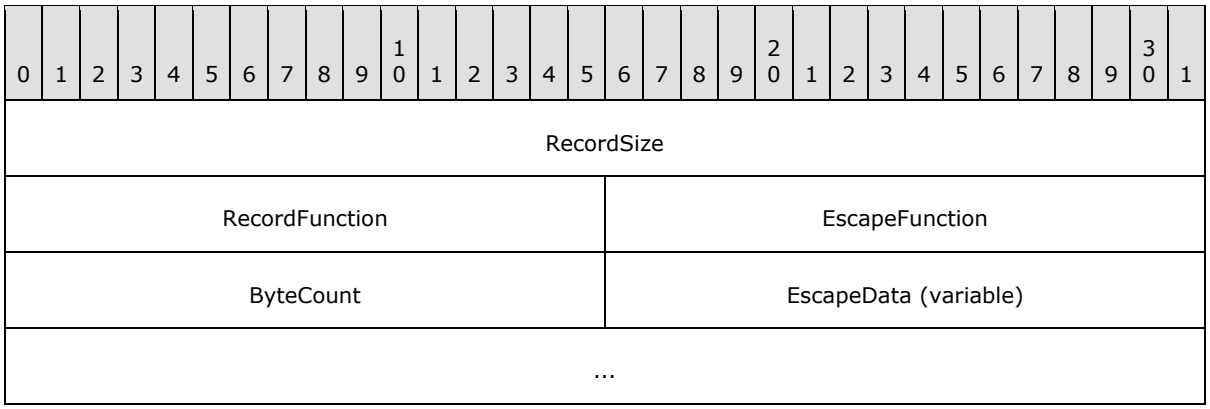
Name	Section	Description
ABORTDOC	2.3.6.2	Stops processing the current document.
BEGIN_PATH	2.3.6.3	Opens a path.

Name	Section	Description
CHECK_JPEGFORMAT	2.3.6.4	Determines whether or not it can handle the given JPEG image.
CHECK_PNGFORMAT	2.3.6.5	Determines whether or not it can handle the given PNG image.
CLIP_TO_PATH	2.3.6.6	Defines a clip region that is bounded by a path.
CLOSE_CHANNEL	2.3.6.7	Same as ENDDOC.
DOWNLOAD_FACE	2.3.6.8	Sets the font face name on a device.
DOWNLOAD_HEADER	2.3.6.9	Downloads sets of PostScript procedures.
DRAW_PATTERNRECT	2.3.6.10	Draws a rectangle with a defined pattern.
ENCAPSULATED_POSTSCRIPT	2.3.6.11	Sends arbitrary encapsulated PostScript (EPS) data directly to the printer driver.
ENDDOC	2.3.6.12	Notifies the printer driver that a new print job is ending.
END_PATH	2.3.6.13	Ends a path.
EPS_PRINTING	2.3.6.14	Indicates the start and end of EPS printing.
EXTTEXTOUT	2.3.6.15	Draws text using the currently selected font, background color, and text color.
GET_PS_FEATURESETTING	2.3.6.16	Queries the printer driver for information about PostScript features supported on the output device.
GET_COLORTABLE	2.3.6.17	Gets color table values.
GET_DEVICEUNITS	2.3.6.18	Gets the device units currently configured on a device.
GET_FACENAME	2.3.6.19	Gets the font face name currently configured on a device.
GET_PAIRKERNTABLE	2.3.6.20	Gets the font kern table currently defined on an output device.
GET_PHYSPAGESIZE	2.3.6.21	Retrieves the physical page size currently selected on an output device.
GET_PRINTINGOFFSET	2.3.6.22	Retrieves the offset from the upper-left corner of the physical page where the actual printing or drawing begins.
GET_SCALINGFACTOR	2.3.6.23	Retrieves the scaling factors for the x-axis and the y-axis of a printer.
GET_EXTENDED_TEXTMETRICS	2.3.6.24	Gets extended text metrics currently configured on an output device.
METAFILE_DRIVER	2.3.6.25	Queries a printer driver about the support for metafiles on an output device.
NEWFRAME	2.3.6.26	Notifies the printer driver that the application has finished writing to a page.

Name	Section	Description
NEXTBAND	2.3.6.27	Notifies the printer driver that the application has finished writing to a band.
PASSTHROUGH	2.3.6.28	This record passes through arbitrary data.
POSTSCRIPT_DATA	2.3.6.29	Sends arbitrary PostScript data to an output device.
POSTSCRIPT_IDENTIFY	2.3.6.30	Sets the printer driver to either PostScript-centric or GDI-centric mode.
POSTSCRIPT_IGNORE	2.3.6.31	Notifies an output device to ignore PostScript data.
POSTSCRIPT_INJECTION	2.3.6.32	Inserts a block of raw data into a PostScript stream.
POSTSCRIPT_PASSTHROUGH	2.3.6.33	Sends arbitrary data directly to a printer driver, which is expected to process this data only when in PostScript mode.
OPEN_CHANNEL	2.3.6.34	The same as **STARTDOC**, with a NULL document and output filename, and data in raw mode.
QUERY_DIBSUPPORT	2.3.6.35	Queries the printer driver about its support for DIBs on an output device.
QUERY_ESCSUPPORT	2.3.6.36	Queries a printer driver to determine whether a specific escape function is supported on the output device it drives.
SET_COLORTABLE	2.3.6.37	Sets color table values.
SET_COPYCOUNT	2.3.6.38	Sets the number of copies.
SET_LINECAP	2.3.6.39	Specifies the line-ending mode to use in output to a device.
SET_LINEJOIN	2.3.6.40	Specifies the line-joining mode to use in output to a device.
SET_MITERLIMIT	2.3.6.41	Sets the limit for the length of miter joins to use in output to a device.
SPCLPASSTHROUGH2	2.3.6.42	Enables documents to include private procedures and other arbitrary data in documents.
STARTDOC	2.3.6.43	Notifies the printer driver that a new print job is starting.
TS_QUERYVER	2.3.6.44	Queries a display driver for Terminal Services support.
TS_RECORD	2.3.6.45	Sends a Terminal Server record to a display device.

2.3.6.1 META_ESCAPE Record

The Windows Metafile Format (WMF) META_ESCAPE record specifies extensions to WMF functionality that are not directly available through other records defined in the WMF **RecordType Enumeration (section 2.1.1)**. The **MetafileEscapes Enumeration (section 2.1.19)** lists these extensions.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** value **META_ESCAPE**.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be one of the values in the **MetafileEscapes Enumeration**.

ByteCount (2 bytes): A 16-bit unsigned integer that defines the number of bytes in the **EscapeData** array.

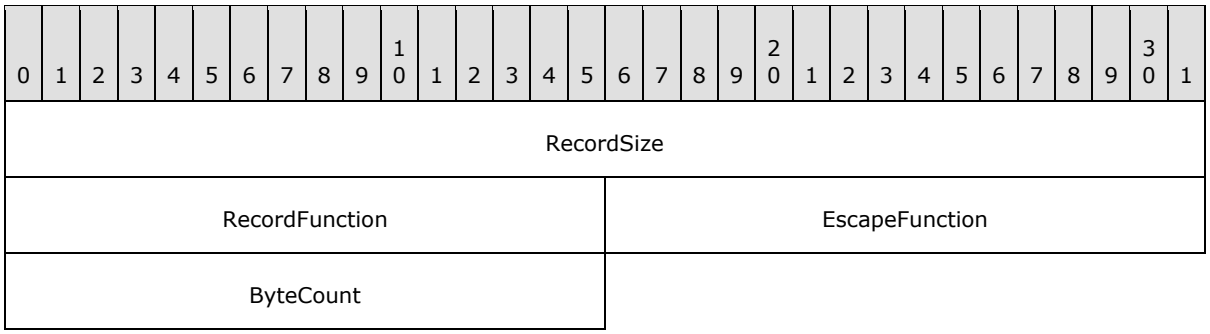
EscapeData (variable): An array of bytes of size **ByteCount**.

Every META_ESCAPE record MUST include a **MetafileEscapes** function specifier, followed by arbitrary data. The data SHOULD NOT contain position-specific data that assumes the location of a particular record within the metafile, because one metafile MAY be embedded within another.

See section 2.3.6 for the specification of other escape record types.

2.3.6.2 ABORTDOC Record

The Windows Metafile Format ABORTDOC record stops processing the current document and erases everything drawn since the last STARTDOC record was processed.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

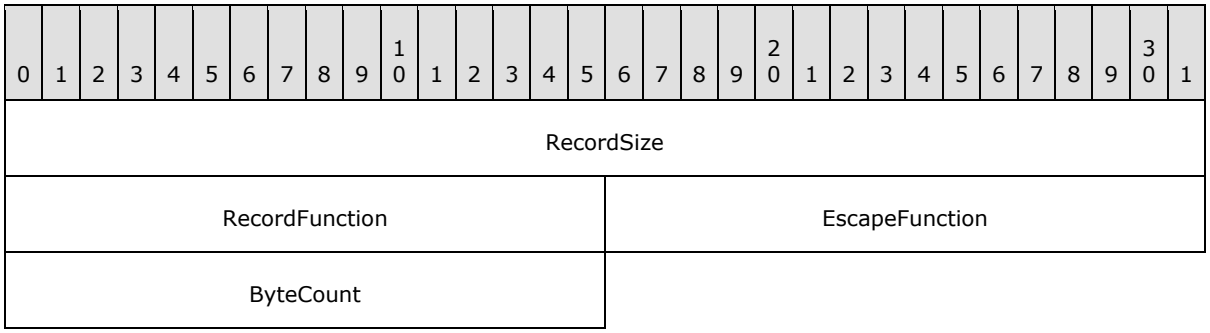
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0002 (ABORTDOC) from **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.3 BEGIN_PATH Record

The Windows Metafile Format BEGIN_PATH record opens a path.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

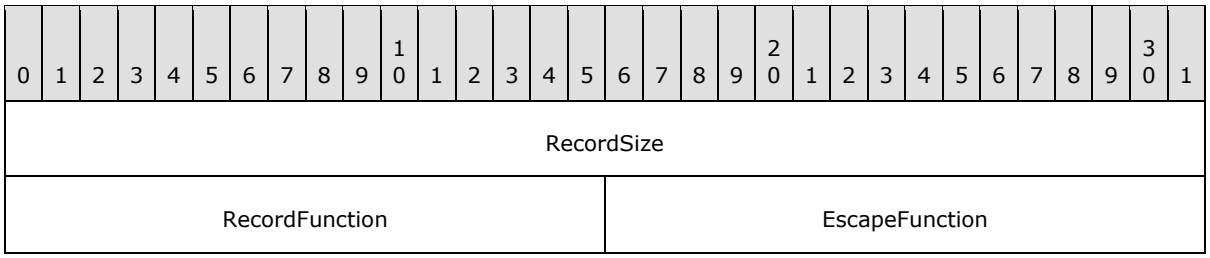
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1000 (BEGIN_PATH) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.4 CHECK_JPEGFORMAT Record

The Windows Metafile Format CHECK_JPEGFORMAT record specifies whether the printer driver supports JPEG image output.



ByteCount	JPEGBuffer (variable)
...	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1007 (CHECK_JPEGFORMAT) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that is size in bytes of the **JPEGBuffer** field.

JPEGBuffer (variable): A variable-sized buffer of JPEG image data.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.5 CHECK_PNGFORMAT Record

The Windows Metafile Format CHECK_PNGFORMAT record queries the driver to see if it can handle the given PNG image and parses the PNG image to determine whether or not the driver can support it.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																EscapeFunction															
ByteCount																PNGBuffer (variable)															
...																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1008 (CHECK_PNGFORMAT) from the **MetafileEscapes Enumeration** table.

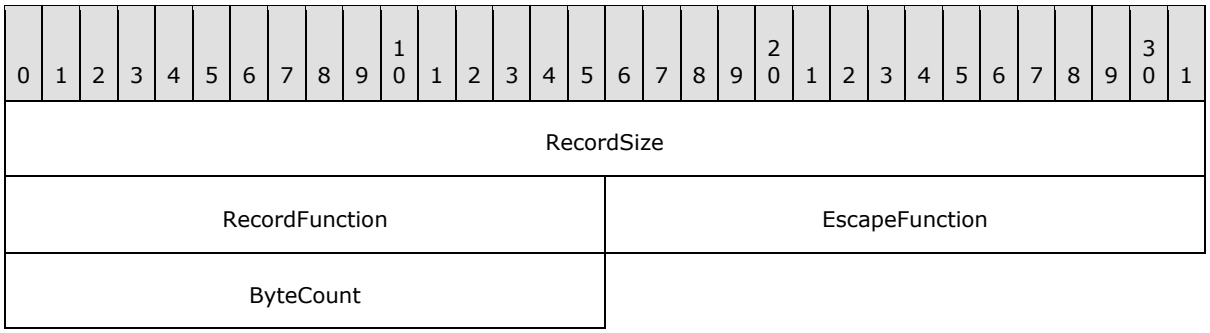
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **PNGBuffer** field.

PNGBuffer (variable): A variable-sized buffer of PNG image data.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.6 CLIP_TO_PATH Record

The Windows Metafile Format CLIP_TO_PATH record defines a clip region that is bounded by a path. The input MUST be a 16-bit quantity that defines the action to take.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

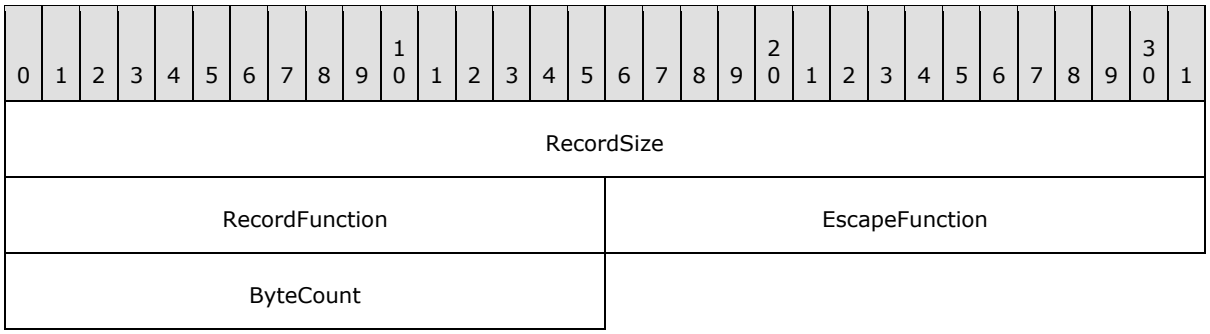
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1001 (CLIP_TO_PATH) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.7 CLOSE_CHANNEL Record

The Windows Metafile Format CLOSE_CHANNEL record is the same as the ENDDOC packet. For more information, see the OPENCHANNEL packet.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

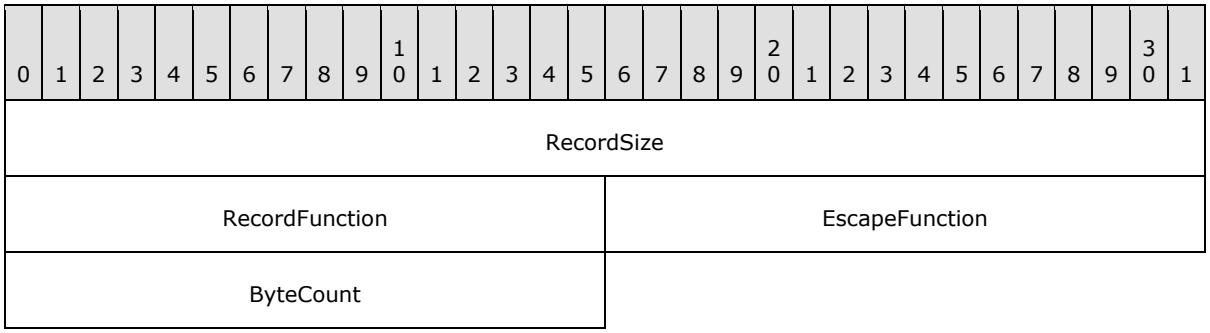
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1012 (CLOSE_CHANNEL) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.8 DOWNLOAD_FACE Record

The Windows Metafile Format DOWNLOAD_FACE record sends the font face.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

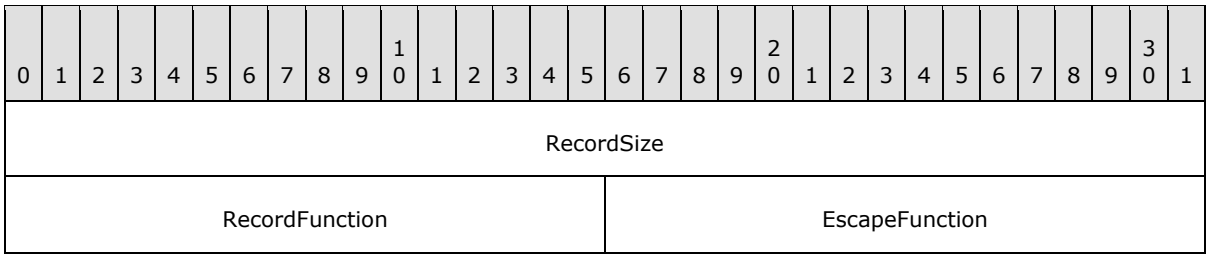
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0202 (DOWNLOAD_FACE) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.9 DOWNLOAD_HEADER Record

The Windows Metafile Format DOWNLOAD_HEADER record instructs the driver to download all sets of PostScript procedures.



ByteCount	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be set to 0x1011 (DOWNLOAD_HEADER) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.10 DRAW_PATTERNRECT Record

The Windows Metafile Format DRAW_PATTERNRECT record draws a rectangle with a defined pattern.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																EscapeFunction															
ByteCount																Position															
...																															
...																Size															
...																															
...																Style															
Pattern																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0019 (DRAW_PATTERNRECT) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that defines the number of bytes in the data that follows in the record.

Position (8 bytes): A 64-bit PointL Object that defines the position of the rectangle.

Size (8 bytes): A 64-bit PointL Object that defines the dimensions of the rectangle.

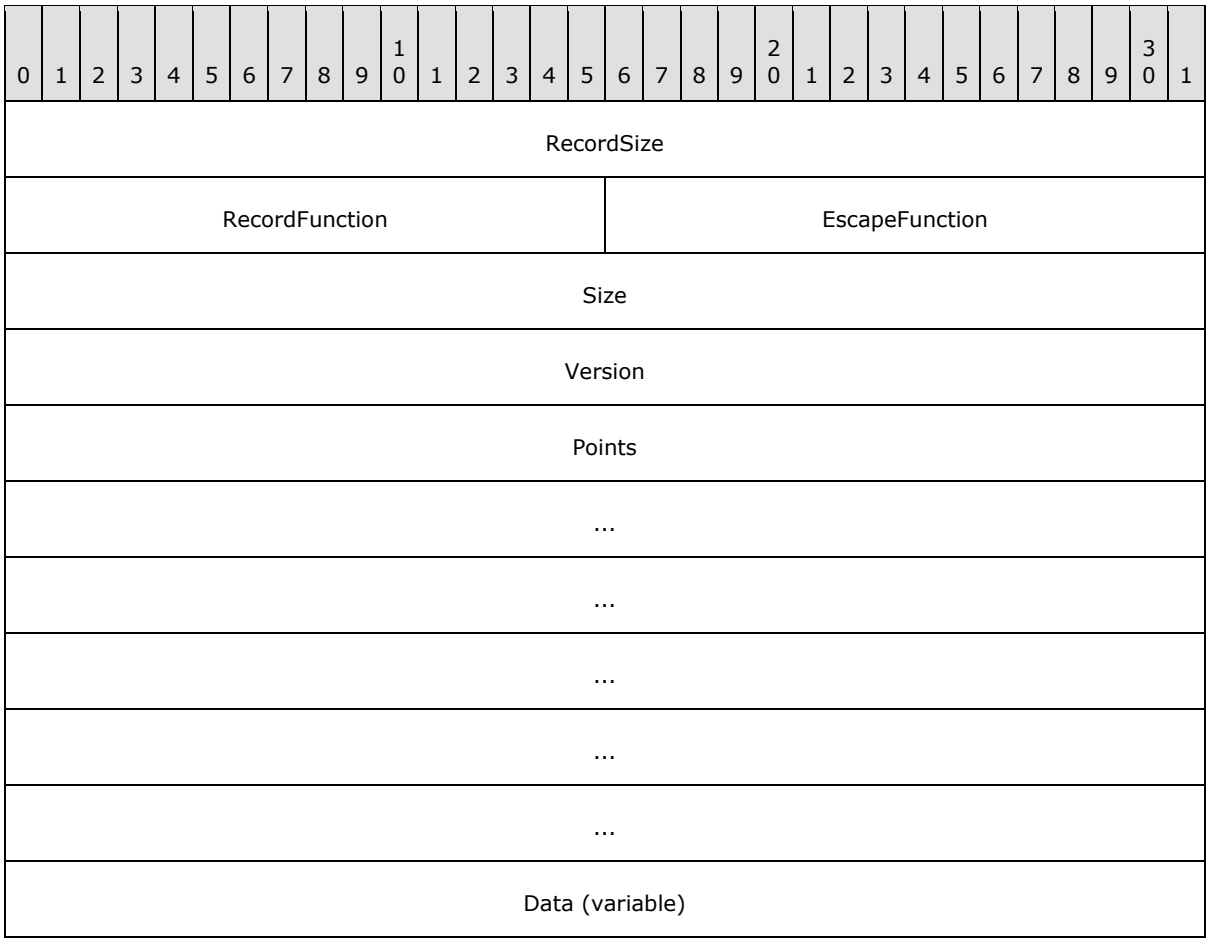
Style (2 bytes): A 16-bit unsigned integer that defines the style.

Pattern (2 bytes): A 16-bit unsigned integer that defines the pattern.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.11 ENCAPSULATED_POSTSCRIPT Record

The Windows Metafile Format ENCAPSULATED_POSTSCRIPT record sends arbitrary data directly to a printer driver.



...

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1014 (ENCAPSULATED_POSTSCRIPT) from the **MetafileEscapes Enumeration** table.

Size (4 bytes): A 32-bit unsigned integer that defines the size of the data including **Size**, **Version**, and **Points**.

Version (4 bytes): A 32-bit unsigned integer that defines the PostScript language level.

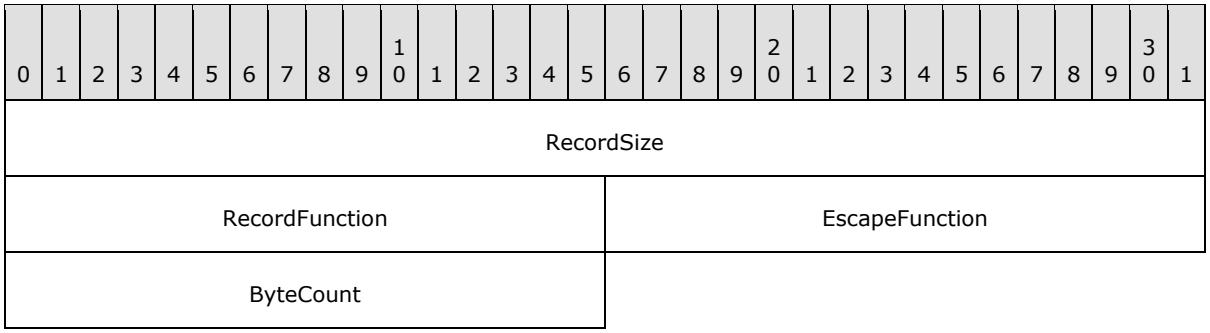
Points (24 bytes): An array of three 64-bit PointL Objects that define the output parallelogram in **28.4 FIX** device coordinates.

Data (variable): The encapsulated data.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.12 ENDDOC Record

The Windows Metafile Format ENDDOC record informs a printer driver that a new print job is ending.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

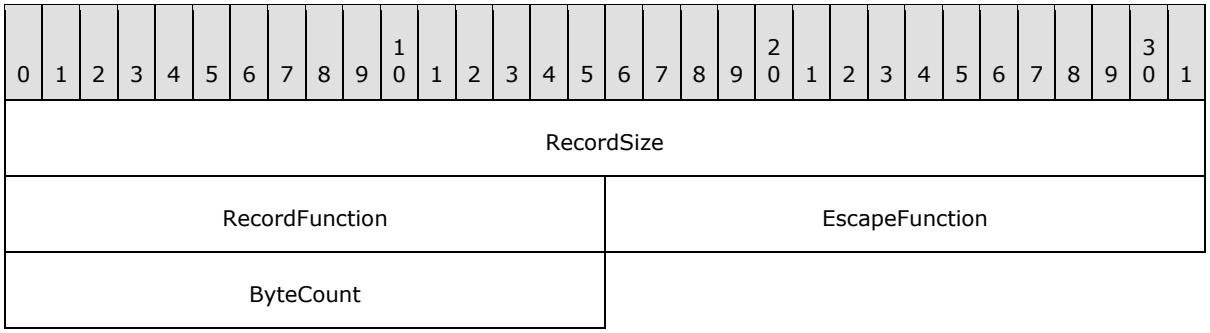
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x000B (ENDDOC) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.13 END_PATH Record

The Windows Metafile Format END_PATH record specifies the end of a path.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

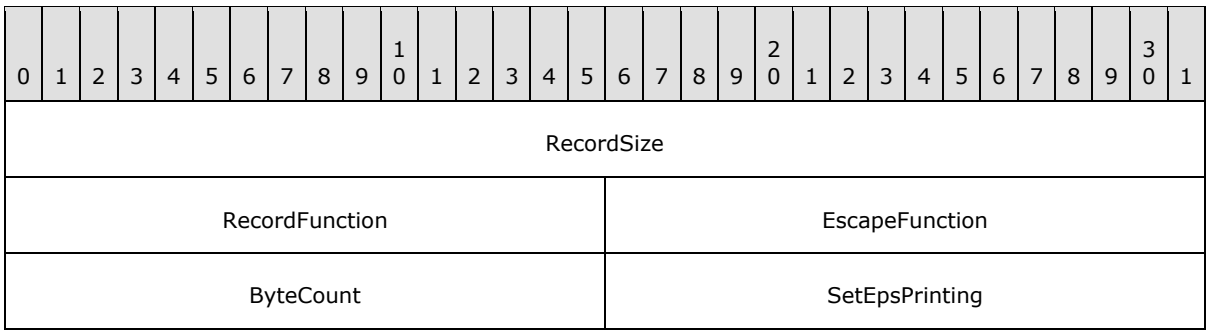
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1002 (END_PATH) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.14 EPS_PRINTING Record

The Windows Metafile Format EPS_PRINTING record indicates the start and the end of encapsulated PostScript printing.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0021 (EPS_PRINTING) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **SetEpsPrinting** field. This MUST be 0x0002.

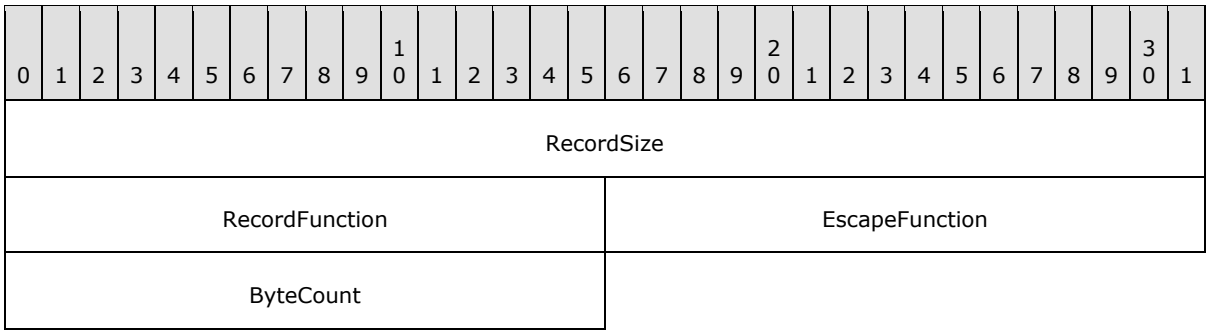
SetEpsPrinting (2 bytes): A 16-bit Boolean value that indicates the start or end of EPS printing. If the value is set, the start of EPS printing is indicated; otherwise, the end is indicated.

Value	Meaning
0	The value is cleared.
0 <*value*	The value is set.
value < 0	The value is set.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.15 EXTTEXTOUT Record

The Windows Metafile Format EXTTEXTOUT record draws text using the currently selected font, background color, and text color.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

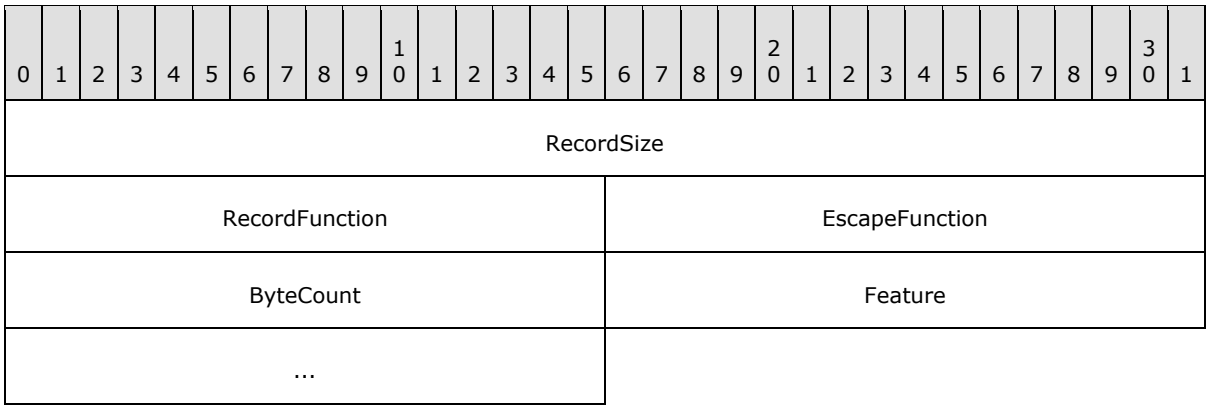
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0200 (EXTTEXTOUT) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.16 GET_PS_FEATURESETTING Record

The Windows Metafile Format GET_PS_FEATURESETTING record is used to query the driver concerning various PostScript features.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1009 (GET_PS_FEATURESETTING) from the **MetafileEscapes Enumeration** table.

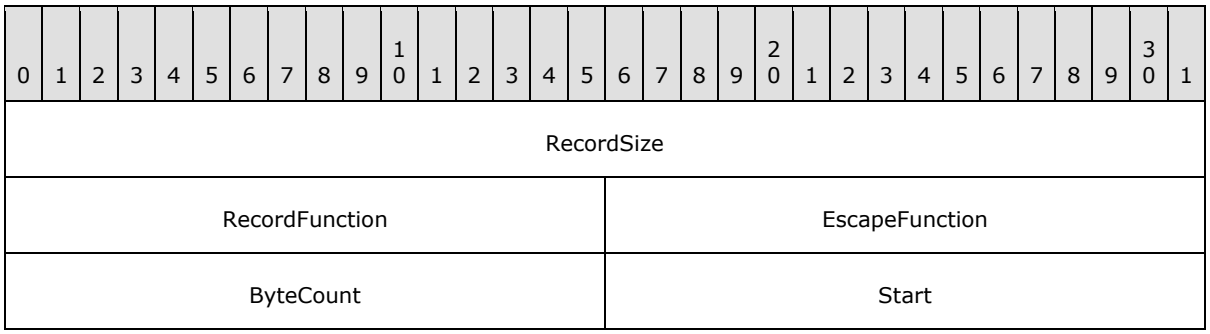
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **Feature** field. This MUST be 0x0004.

Feature (4 bytes): A 32-bit signed integer that identifies the feature setting being queried. Possible values are defined in the **PostScriptFeatureSetting Enumeration** table.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.17 GET_COLORTABLE Record

The Windows Metafile Format GET_COLORTABLE record gets the color table.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0005 (GET_COLORTABLE) from the **MetafileEscapes Enumeration** table.

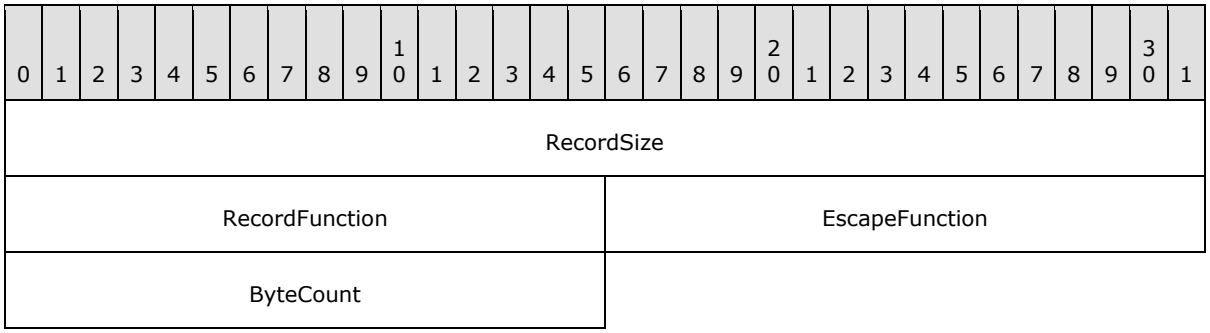
ByteCount (2 bytes): A 16-bit unsigned integer that defines the number of bytes in the color table.

Start (2 bytes): A 16-bit unsigned integer that defines the offset to the beginning of the color table data.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.18 GET_DEVICEUNITS Record

The Windows Metafile Format GET_DEVICEUNITS record gets the current device units.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

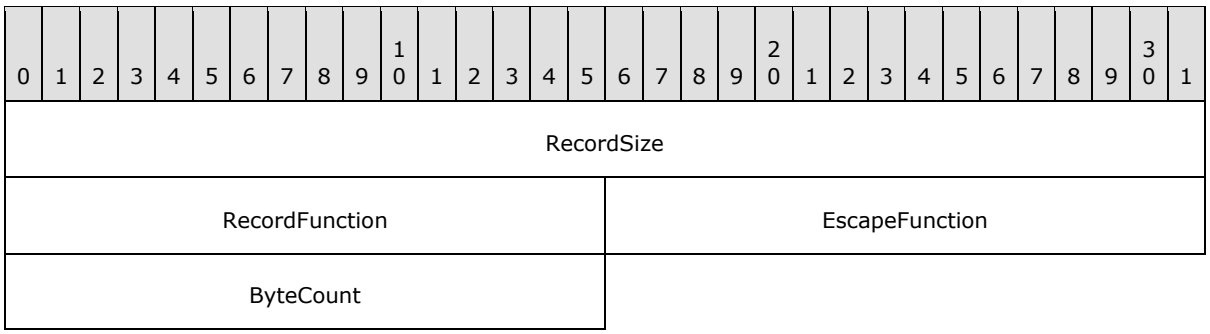
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x002A (GET_DEVICEUNITS) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.19 GET_FACENAME Record

The Windows Metafile Format GET_FACENAME record gets the font face name.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

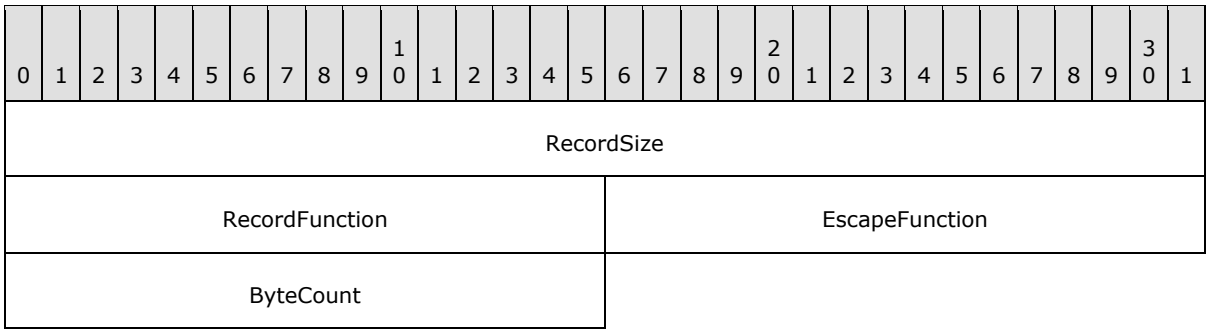
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0201 (GET_FACENAME) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.20 GET_PAIRKERNTABLE Record

The Windows Metafile Format GET_PAIRKERNTABLE record gets the font kern table.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

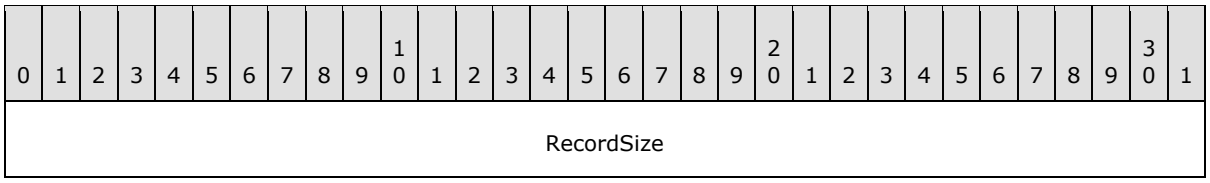
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0102 (GET_PAIRKERNTABLE) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.21 GET_PHYSPAGESIZE Record

The Windows Metafile Format GET_PHYSPAGESIZE record retrieves the physical page size and copies it to a specified location.



RecordFunction	EscapeFunction
ByteCount	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x000C (GET_PHYSPAGESIZE) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.22 GET_PRINTINGOFFSET Record

The Windows Metafile Format GET_PRINTINGOFFSET record retrieves the offset from the upper-left corner of the physical page where the actual printing or drawing begins.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">RecordSize</td></tr>
<tr><td colspan="16">RecordFunction</td><td colspan="16">EscapeFunction</td></tr>
<tr><td colspan="16">ByteCount</td><td colspan="16"></td></tr>
</table>

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

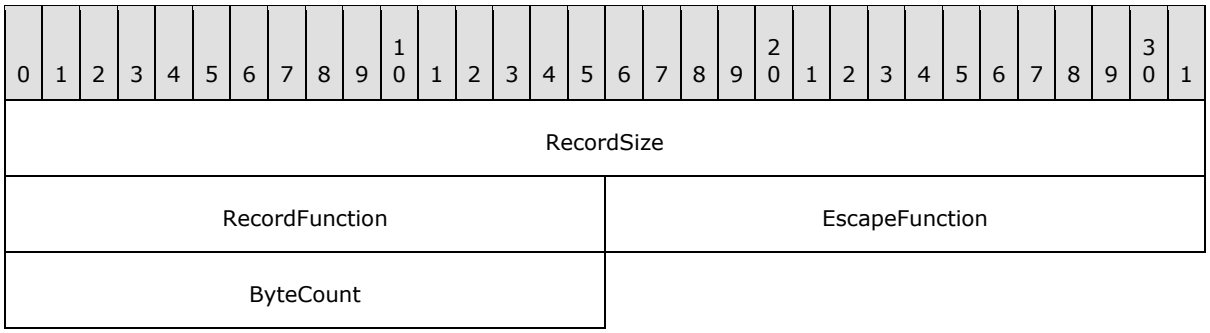
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x000D (GET_PRINTINGOFFSET) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.23 GET_SCALINGFACTOR Record

The Windows Metafile Format GET_SCALINGFACTOR record retrieves the scaling factors for the x-axis and the y-axis of a printer.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

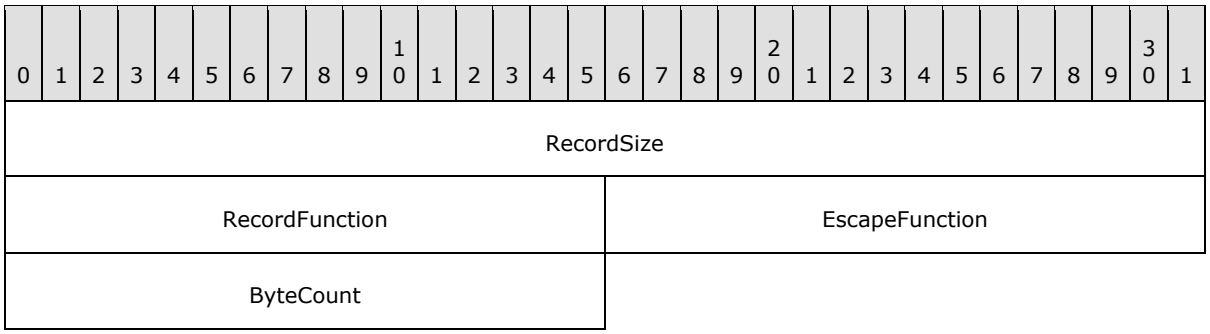
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x000E (GET_SCALINGFACTOR) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.24 GET_EXTENDED_TEXTMETRICS Record

The Windows Metafile Format GET_EXTENDED_TEXTMETRICS record gets the extended text metrics.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

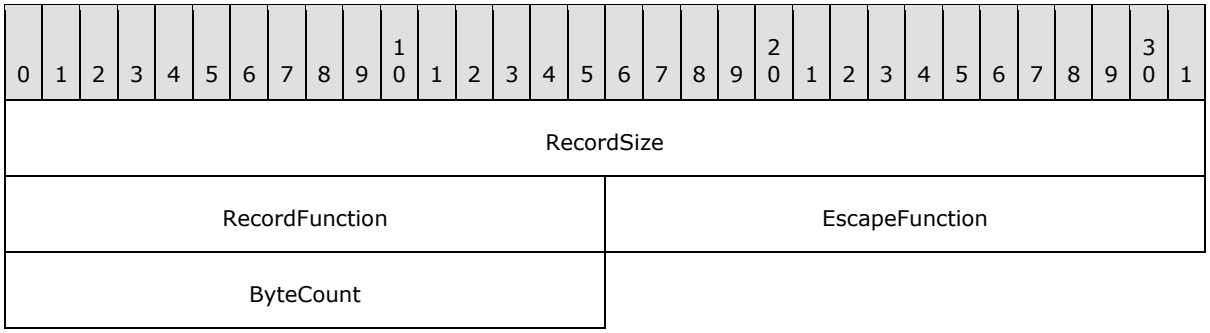
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0100 (GET_EXTENDED_TEXTMETRICS) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.25 METAFILE_DRIVER Record

The Windows Metafile Format METAFILE_DRIVER record queries the driver about its support for metafiles.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

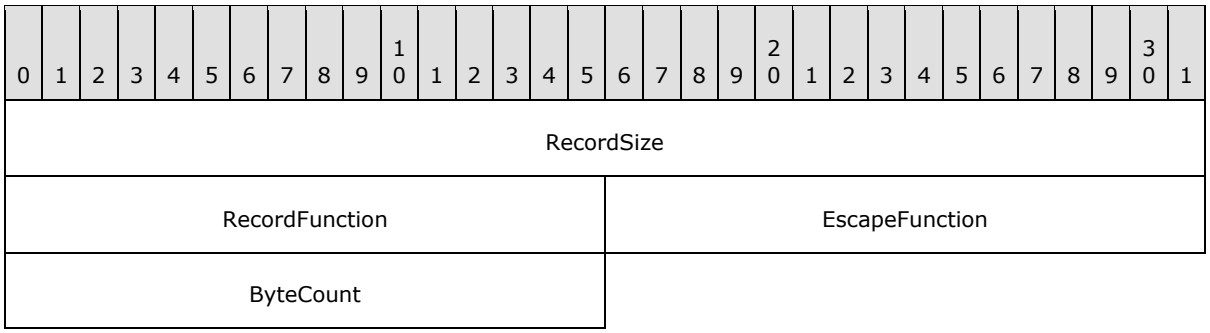
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0801 (METAFILE_DRIVER) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.26 NEWFRAME Record

The Windows Metafile Format NEWFRAME record informs the printer that the application has finished writing to a page.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0001 (NEWFRAME) from the **MetafileEscapes Enumeration** table.

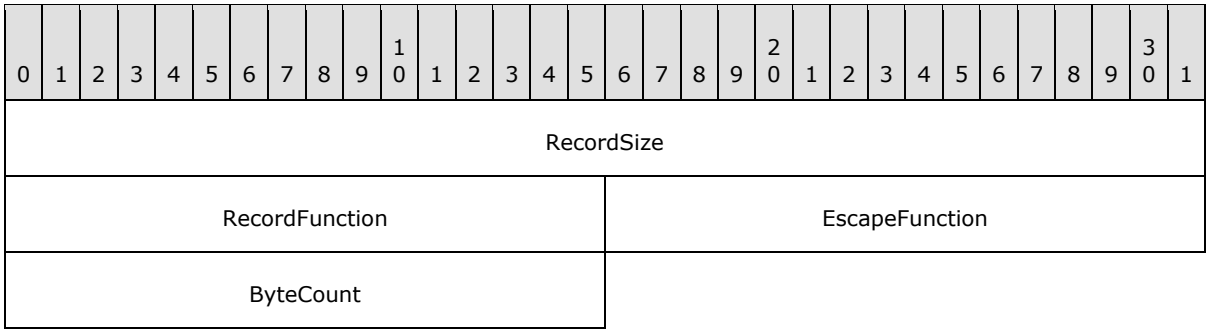
ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.27 NEXTBAND Record

The Windows Metafile Format NEXTBAND record informs the printer that the application has finished writing to a band.

Band information is no longer used.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

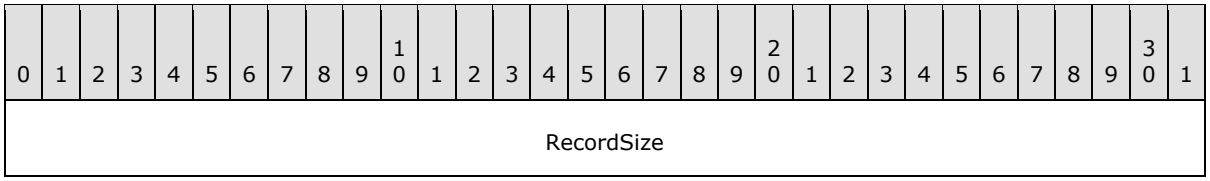
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0003 (NEXTBAND) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.28 PASSTHROUGH Record

The Windows Metafile Format (WMF) PASSTHROUGH record passes through arbitrary data.



RecordFunction	EscapeFunction
ByteCount	Data (variable)
...	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value for the META_ESCAPE (section 2.3.6.1) record.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0013 (PASSTHROUGH) from the **MetafileEscapes Enumeration (section 2.1.19)**.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the number of bytes in the **Data** field.

Data (variable): An array of bytes of size **ByteCount**.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.29 POSTSCRIPT_DATA Record

The Windows Metafile Format (WMF) POSTSCRIPT_DATA record sends arbitrary PostScript data within the record.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																EscapeFunction															
ByteCount																Data (variable)															
...																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value META_ESCAPE (section 2.3.6.1) record.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0025 (POSTSCRIPT_DATA) from the **MetafileEscapes Enumeration (section 2.1.19)**.

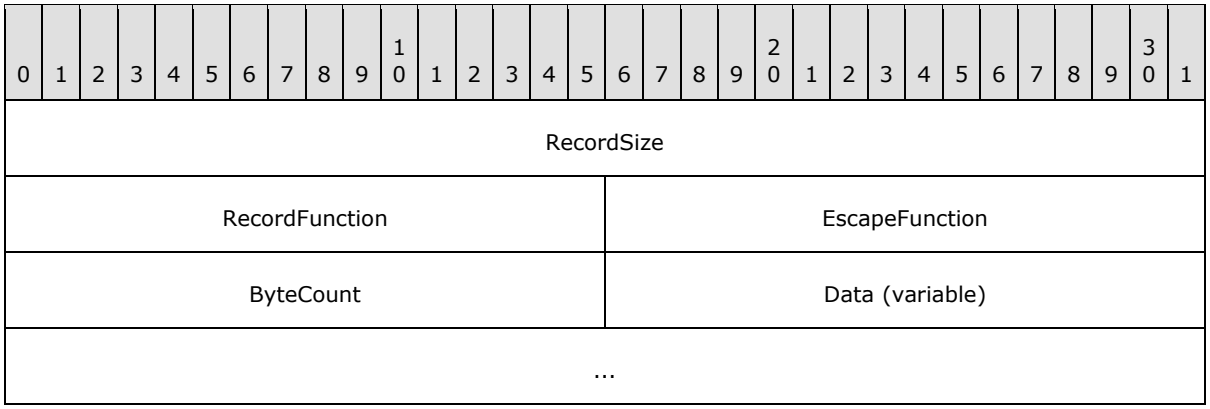
ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the number of bytes in the **Data** field.

Data (variable): An array of bytes of size **ByteCount**.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.30 POSTSCRIPT_IDENTIFY Record

The Windows Metafile Format (WMF) POSTSCRIPT_IDENTIFY record sets the driver to either PostScript-centric or GDI-centric mode.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value META_ESCAPE (section 2.3.6.1) record.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1005 (POSTSCRIPT_IDENTIFY) from the **MetafileEscapes Enumeration (section 2.1.19)**.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the number of bytes in the **Data** field.

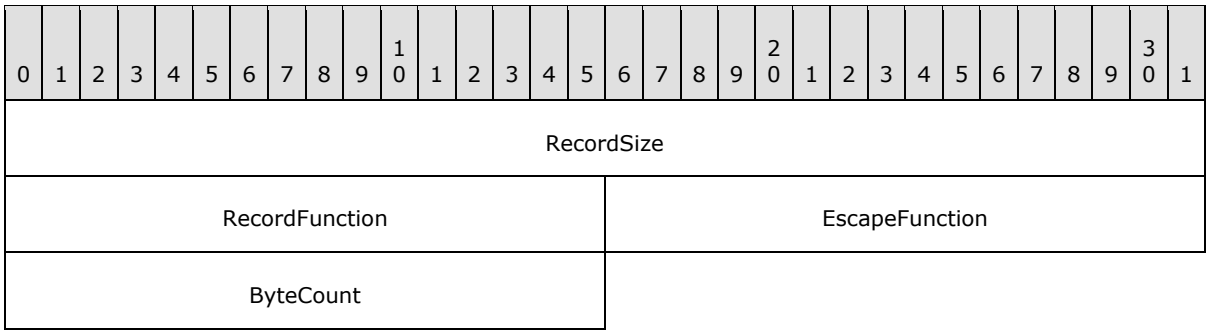
Data (variable): An array of bytes of size **ByteCount**.

Note This record must be processed before the STARTDOC record.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.31 POSTSCRIPT_IGNORE Record

The Windows Metafile Format POSTSCRIPT_IGNORE record informs the device to ignore the PostScript data.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

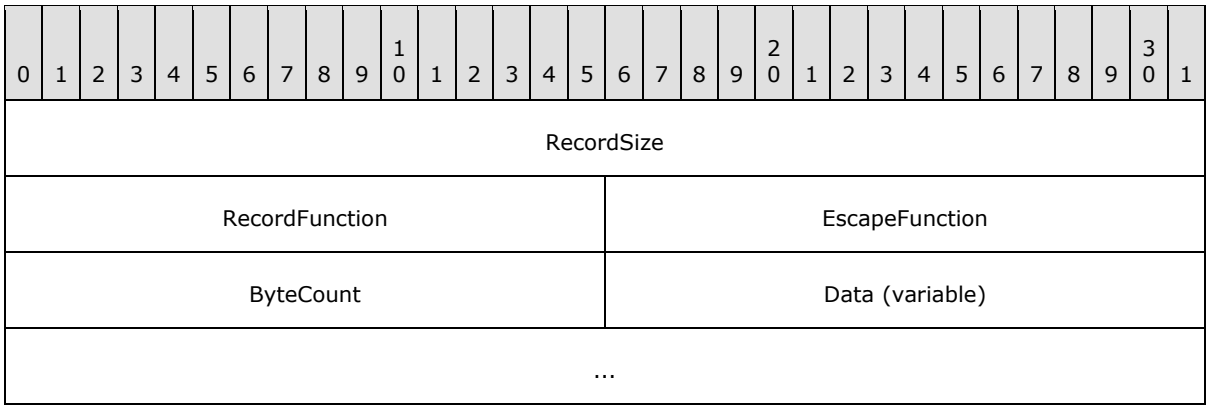
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0026 (POSTSCRIPT_IGNORE) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.32 POSTSCRIPT_INJECTION Record

The Windows Metafile Format (WMF) POSTSCRIPT_INJECTION record inserts a block of raw data into a PostScript stream. The input MUST be a 32-bit quantity specifying the number of bytes to inject, a 16-bit quantity specifying the injection point, and a 16-bit quantity specifying the page number, followed by the bytes to inject.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value META_ESCAPE (section 2.3.6.1) record.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1006 (POSTSCRIPT_INJECTION) from the **MetafileEscapes Enumeration (section 2.1.19)**.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the number of bytes in the **Data** field.

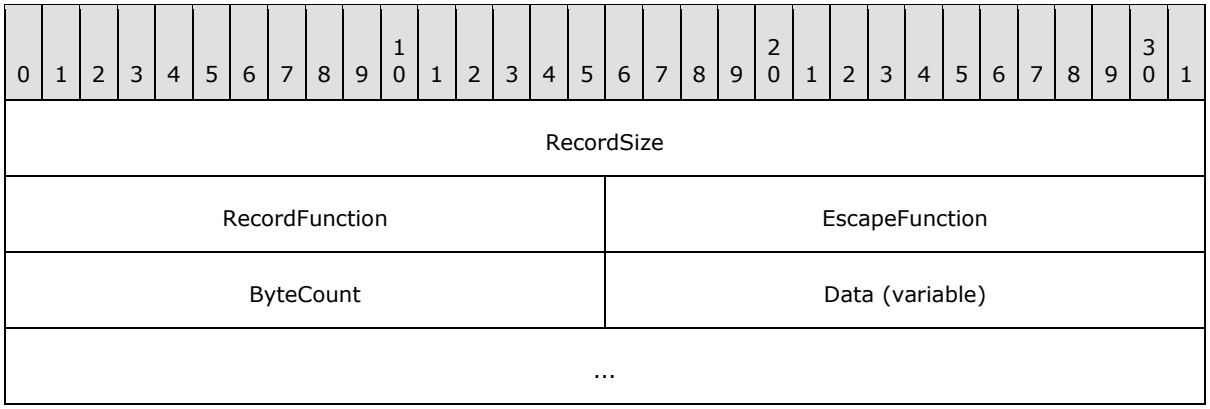
Data (variable): An array of bytes of size **ByteCount**.

Note This record must be processed before a STARTDOC (section 2.3.6.43) record.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.33 POSTSCRIPT_PASSTHROUGH Record

The Windows Metafile Format (WMF) POSTSCRIPT_PASSTHROUGH record sends arbitrary data directly to the driver. The driver is expected to only process this data when in PostScript mode. For more information, see the POSTSCRIPT_IDENTIFY (section 2.3.6.30) Escape record.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value META_ESCAPE (section 2.3.6.1).

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1013 (POSTSCRIPT_PASSTHROUGH) from the **MetafileEscapes Enumeration (section 2.1.19)**.

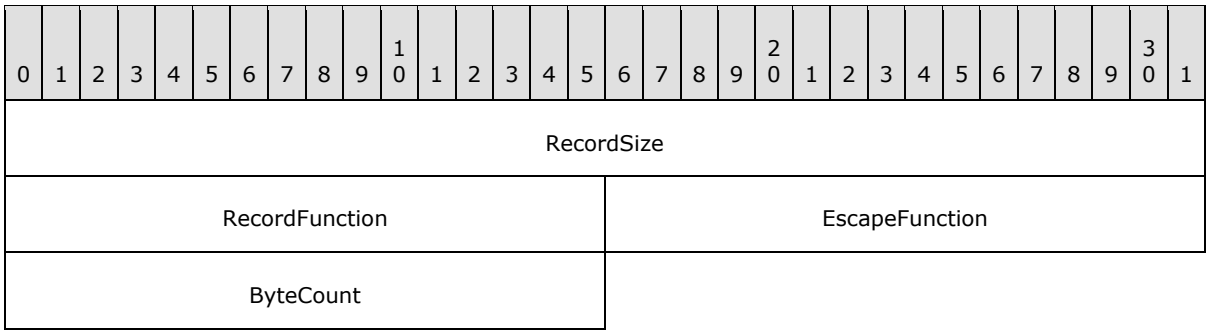
ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the number of bytes in the **Data** field.

Data (variable): An array of bytes of size **ByteCount**.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.34 OPEN_CHANNEL Record

The Windows Metafile Format OPEN_CHANNEL record is a synonym for STARTDOC with a NULL document and output file name, a datatype of RAW, and a type of 0.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

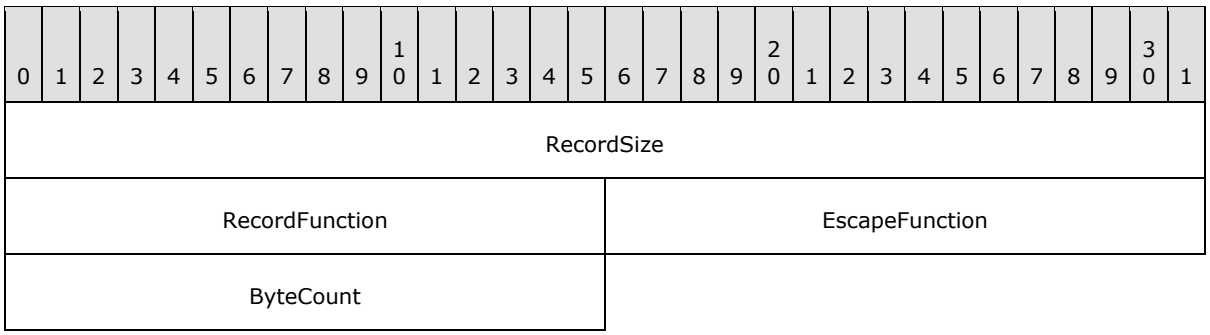
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x1010 (OPEN_CHANNEL) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.35 QUERY_DIBSUPPORT Record

The Windows Metafile Format QUERY_DIBSUPPORT record queries the driver about its support for DIBs.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

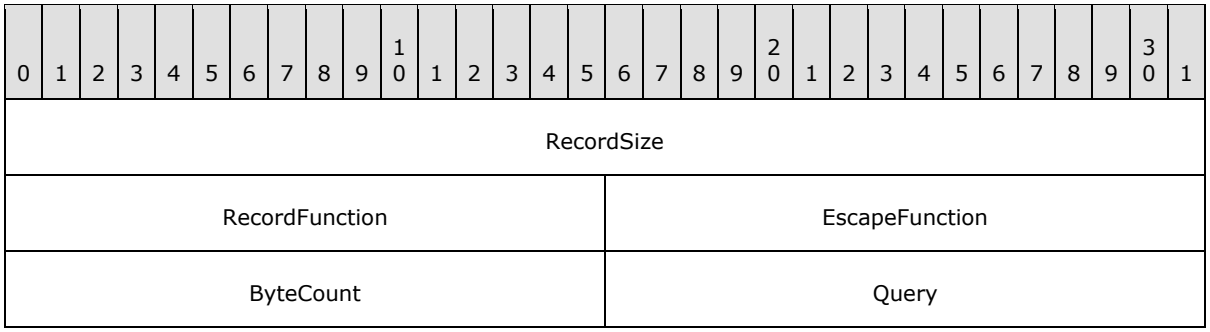
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0C01 (QUERY_DIBSUPPORT) from the **MetafileEscapes Enumeration** table.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.36 QUERY_ESCSUPPORT Record

The Windows Metafile Format QUERY_ESCSUPPORT record queries record to send to the device to determine whether a specific escape function is supported.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0008 (QUERY_ESCSUPPORT) from the **MetafileEscapes Enumeration** table.

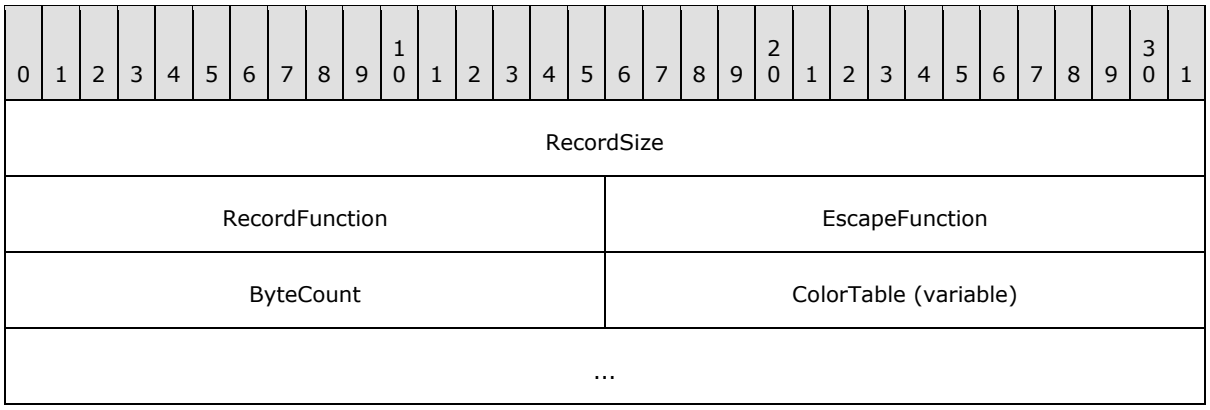
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **Query** field. This MUST be 0x0002.

Query (2 bytes): A 16-bit unsigned integer that MUST come from the **MetafileEscapes Enumeration** table. This will check whether there is support for this escape.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.37 SET_COLORTABLE Record

The Windows Metafile Format (WMF) SET_COLORTABLE record sets color table.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0004 (SET_COLORTABLE) from the **MetafileEscapes Enumeration (section 2.1.19)** table.

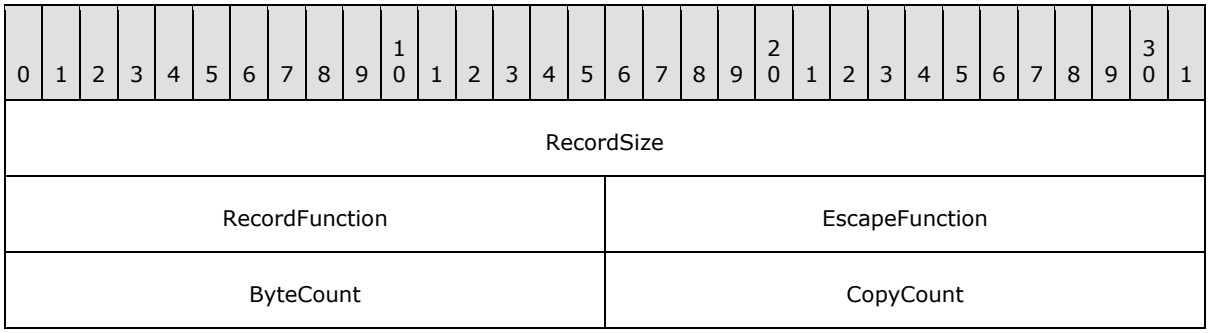
ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the number of bytes in **ColorTable**.

ColorTable (variable): A **ByteCount** length byte array containing the color table.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.38 SET_COPYCOUNT Record

The Windows Metafile Format SET_COPYCOUNT record sets the number of copies.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0011 (SET_COPYCOUNT) from the **MetafileEscapes Enumeration** table.

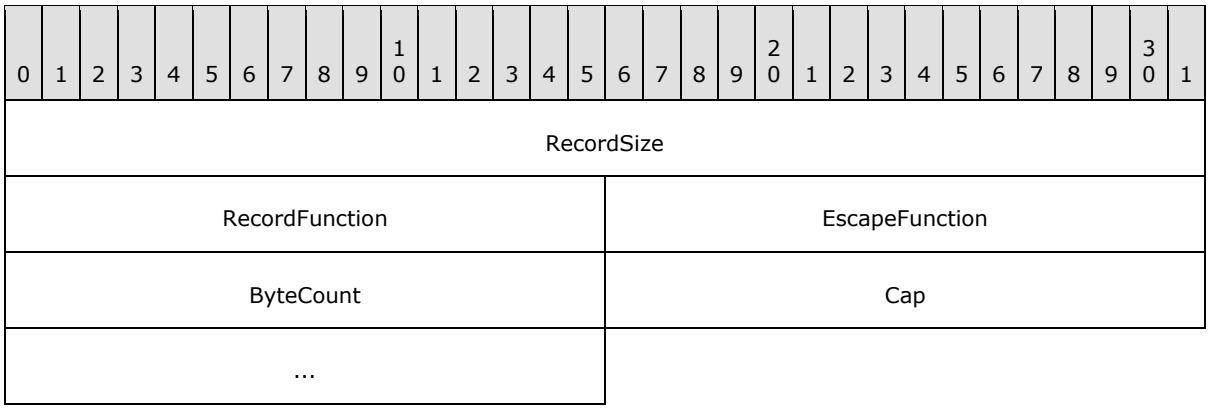
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **CopyCount** field. This MUST be 0x0002.

CopyCount (2 bytes): A 16-bit unsigned integer that specifies the number of copies to print.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.39 SET_LINECAP Record

The Windows Metafile Format SET_LINECAP record specifies the type of line-ending to use in subsequent graphics operations.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The low-order byte MUST match the low-order byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0015 (SET_LINECAP) from the **MetafileEscapes Enumeration** table.

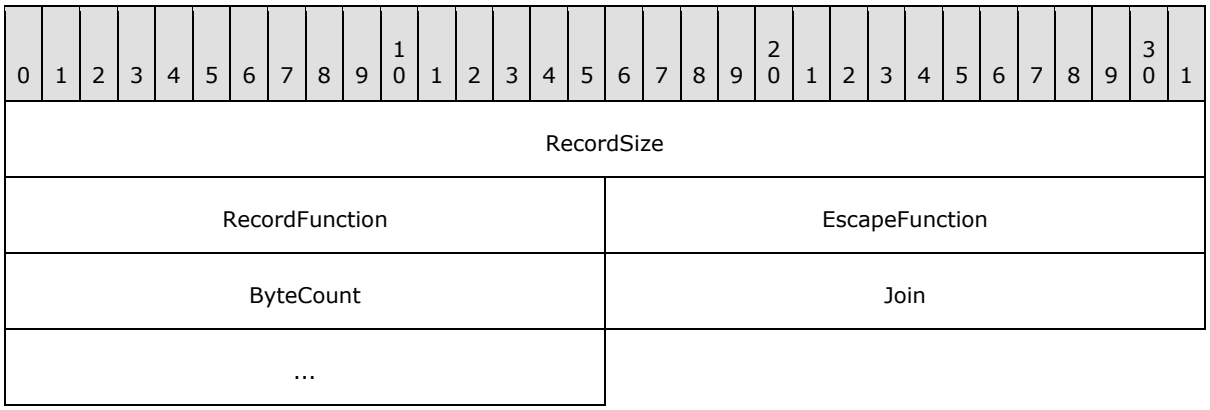
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **Cap** field. This MUST be 0x0004.

Cap (4 bytes): A 32-bit signed integer that defines the type of line cap. Possible values are specified in the **PostScriptCap Enumeration** table.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.40 SET_LINEJOIN Record

The Windows Metafile Format SET_LINEJOIN record specifies the type of line-joining to use in subsequent graphics operations.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this WMF record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0016 (SET_LINEJOIN) from the **MetafileEscapes Enumeration** table.

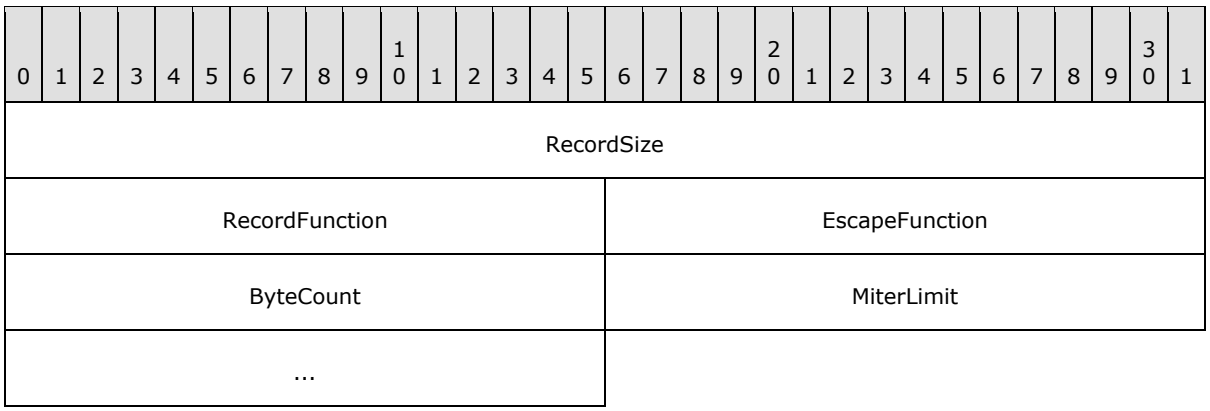
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **Join** field. This MUST be 0x0004

Join (4 bytes): A 32-bit signed integer that specifies the type of line join. Possible values are specified in **PostScriptJoin Enumeration** table.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.41 SET_MITERLIMIT Record

The Windows Metafile Format SET_MITERLIMIT record sets the limit for the length of miter joins for the specified device context.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x0017 (SET_MITERLIMIT) from the **MetafileEscapes Enumeration** table.

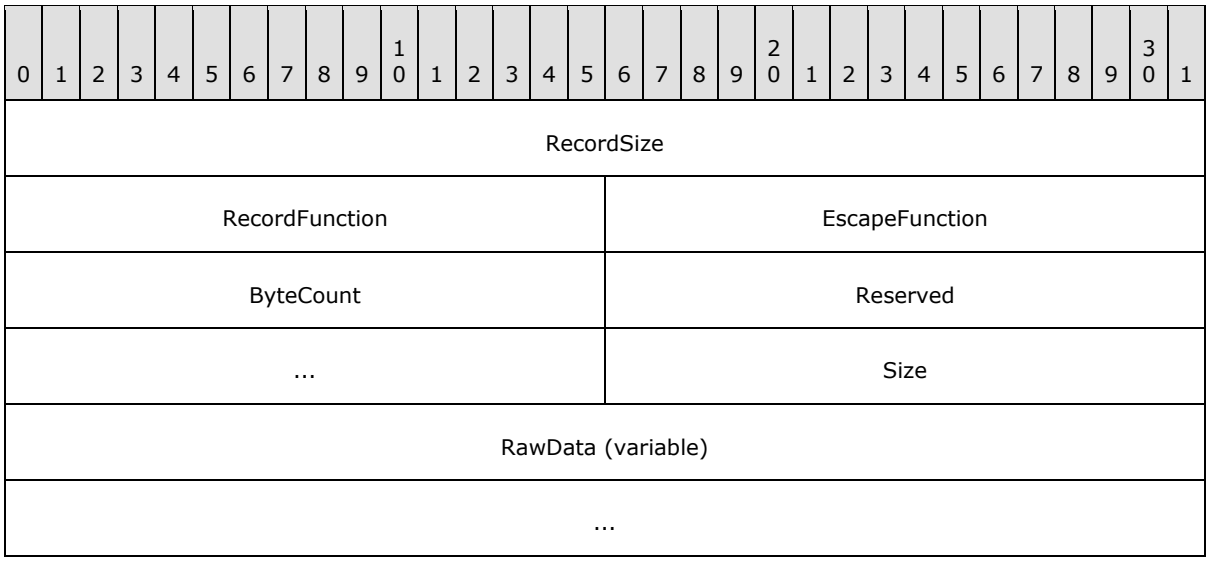
ByteCount (2 bytes): A 16-bit unsigned integer that is the size, in bytes, of the **MiterLimit** field. This MUST be 0x0004.

MiterLimit (4 bytes): A 32-bit signed integer that specifies the miter limit.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.42 SPCLPASSTHROUGH2 Record

The Windows Metafile Format (WMF) SPCLPASSTHROUGH2 record enables documents to include private procedures and other resources at the document level-save context.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x11D8 (SPCLPASSTHROUGH2) from the **MetafileEscapes Enumeration (section 2.1.19)** table.

ByteCount (2 bytes): A 16-bit unsigned integer that is the size in bytes of all the fields following this one in the record.

Reserved (4 bytes): A 32-bit unsigned integer that is not used and MUST be ignored.

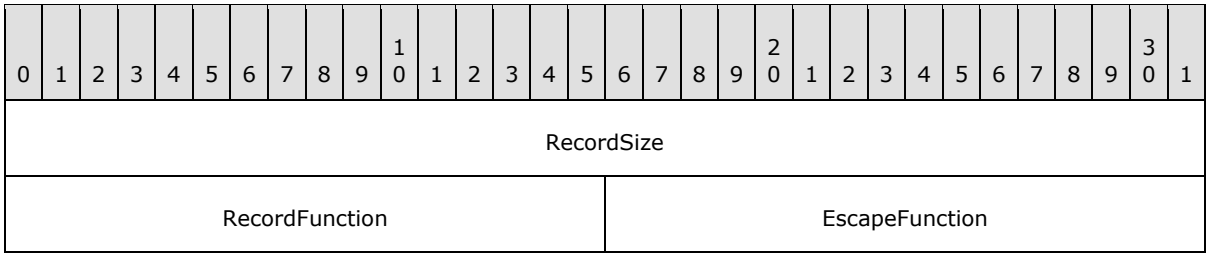
Size (2 bytes): A 16-bit unsigned integer that is the size in bytes of the **RawData** array.

RawData (variable): The **Size** bytes of unprocessed private data to send to the printer driver.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.43 STARTDOC Record

The Windows Metafile Format (WMF) STARTDOC record informs a printer driver that a new print job is starting.



ByteCount	DocName (variable)
...	

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** table value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x000A (STARTDOC) from the **MetafileEscapes Enumeration (section 2.1.19)** table.

ByteCount (2 bytes): A 16-bit unsigned integer that specifies the number of bytes in the **DocName** field. This size MUST be less than 260 characters long.

DocName (variable): A string of **ByteCount** 8-bit characters that contains the name of the document.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.44 TS_QUERYVER Record

The Windows Metafile Format (WMF) TS_QUERYVER record queries a device for Terminal Services support.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordSize																															
RecordFunction																EscapeFunction															
ByteCount																															

RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value META_ESCAPE.

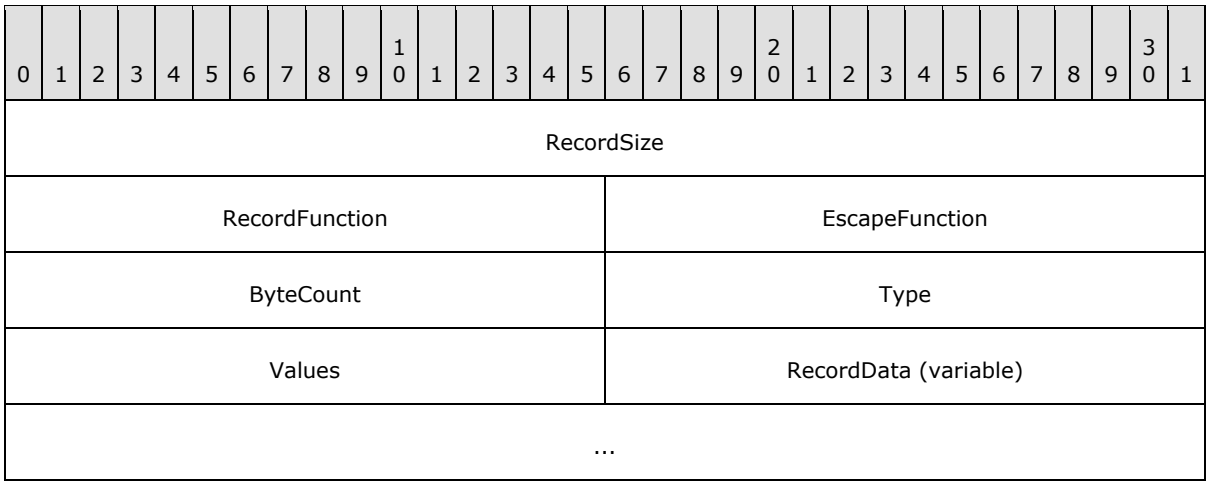
EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x100A (TS_QUERYVER) from the **MetafileEscapes Enumeration (section 2.1.19)**.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be 0x0000.

See section 2.3.6 for the specification of other Escape record types.

2.3.6.45 TS_RECORD Record

The Windows Metafile Format (WMF) TS_RECORD record sends a record to a device with Terminal Services support.



RecordSize (4 bytes): A 32-bit unsigned integer that defines the number of **WORDs** in the WMF record.

RecordFunction (2 bytes): A 16-bit unsigned integer that defines this record type. The lower byte MUST match the lower byte of the **RecordType Enumeration (section 2.1.1)** value META_ESCAPE.

EscapeFunction (2 bytes): A 16-bit unsigned integer that defines the escape function. The value MUST be 0x100B (TS_RECORD) from the **MetafileEscapes Enumeration (section 2.1.19)**.

ByteCount (2 bytes): A 16-bit unsigned integer that MUST be the size, in bytes, of the following record members.

Type (2 bytes): A 16-bit signed integer that defines the type of the record.

Values (2 bytes): A 16-bit unsigned integer that defines the values.

RecordData (variable): Variable-length data that defines the records to be drawn.

See section 2.3.6 for the specification of other Escape record types.

3 Structure Examples

3.1 Metafile Design

3.1.1 Device Independence

Windows Metafile Format (WMF) metafiles are useful for transferring images between applications. Most applications support the Clipboard format associated with metafiles, called **METAFILEPICT** (for more information, see [MSDN-CLIPFORM]). When treated as a single graphics primitive, a metafile is easy to paste into an application without that application needing to know the specific content of the picture. An application can store a metafile in global memory or to disk.

The mapping mode of a metafile can be altered during playback. Thus, the image can be scaled arbitrarily, with every component scaling separately, which minimizes the loss of information for the image as a whole, which is not characteristic of bitmaps.

To ensure that metafiles can be transported between different computers and applications, any application that creates a metafile should make sure the metafile is device independent and sizable. The following guidelines help ensure that every metafile can be accepted and manipulated by other applications.

- Set the mapping mode as one of the first records. Some applications only accept metafiles that are in MM_ANISOTROPIC mode.
- Set the META_SETWINDOWORG and META_SETWINDOWEXT records. Do not use the META_SETVIEWPORTEXT or META_SETVIEWPORTORG record if the user will be able to resize or change the dimensions of the object.
- Use the MFCOMMENT printer escape, specified in section 2.1.19 , to add comments to the metafile.
- Do not use any of the region records, because they are device dependent.
- Use META_STRETCHBLT or META_STRETCHDIB instead of META_BITBLT.
- Terminate the metafile with a META_EOF (0x0000) record.

3.1.2 Byte Ordering Example

The following code snippet illustrates how the use of the big-endian and little-endian methods can affect the compatibility of applications.

```
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
 int buf;
 int in;
 int nread;
 in = open("file.in", O_RDONLY);
 nread = read(in, (int *) &buf, sizeof(buf));
 printf("First Integer in file.in = %x\n", buf);
 exit(0);
}
```

In the preceding code, if the first integer word stored in the file.in file on a big-endian computer was the hexadecimal number 0x12345678, the resulting output on that computer would be as follows:

```
% ./test
First Integer in file.in = 12345678
%
```

If the file.in file were read by the same program running on a little-endian computer, the resulting output would be as follows:

```
% ./test
First Integer in file.in = 78563412
%
```

Because of the difference in output, metafile record processing should be implemented so that it can read integers from a file based on the endian method that the output computer uses.

Because metafiles were developed and written with little-endian computers, computers that are big-endian based will have to perform this necessary compensation.

3.1.3 Mapping Modes

When an application pastes a Windows Metafile Format (WMF) metafile from the clipboard, it can determine the size of metafile output. For this to work cleanly between applications, be aware of the following:

- The metafile is responsible for specifying the window part of the mapping mode; and
- The player of the metafile is responsible for the viewport part of the mapping mode.

To perform a simple playback of the metafile, an application SHOULD perform the following initialization before processing records.

1. Set the mapping mode to the mode specified in the METAFILEPICT structure (for more information, see [MSDN-CLIPFORM]);
2. Convert the horizontal and vertical extents of the viewport to logical units, if necessary (see below);
3. Perform scaling computations, if required (see below); and
4. Set the viewport origin according to the desired placement of the metafile.

If the mapping mode in a metafile is MM_ANISOTROPIC or MM_ISOTROPIC, coordinate conversion MAY be required, as mentioned above. If the horizontal and vertical extents of the image are given in MM_HIMETRIC coordinates, they MUST be converted to pixel values. Before playback, the application needs to set the viewport origin to the desired location, set the mapping mode to the specified mode, and compute the viewport extents. If no extents are specified in the METAFILEPICT structure, the application performing the playback MAY arbitrarily choose a size.

Thus, scaling a metafile that uses the MM_ANISOTROPIC or MM_ISOTROPIC mapping modes MAY be performed by changing the viewport extents to the appropriate dimensions before playback. The viewport defines the size of the metafile image.

To scale metafiles that use any other mapping mode, first convert the metafile to use MM_ANISOTROPIC mapping mode. The metafile itself does not need to change, but the mapping mode setup does need to be correct before beginning the playback.

3.1.4 Managing Objects

3.1.4.1 WMF Object Table

The Windows Metafile Format (WMF) Object Table is a conceptual element of WMF graphics objects management. Graphics objects include Brushes, Fonts, Palettes, Pens and Regions; they can be defined, used, reused and deleted by records in a WMF metafile. This section describes a hypothetical WMF Object Table to keep track of graphics objects during the processing of a WMF metafile.

The WMF Object Table is simply an array of indexes assigned to graphics object structures defined during the processing of a WMF metafile. The maximum number of indexes needed in a WMF Object Table for a given metafile can be computed from the total number of objects defined in the metafile, which is specified by the **NumberOfObjects** field in the WMF META_HEADER record. An implementation of the WMF Object Table MUST be able to store and manage that number of objects.

Whenever a graphics object is created by one of the WMF Object records listed in section 2.3.4 , the following actions are implied.

- Every object MUST be assigned the lowest available index—that is, the smallest numerical value —in the WMF Object Table. This binding MUST happen at object creation; it MUST NOT be deferred until the object is used. Moreover, each object table index MUST uniquely refer to an object. Indexes in the WMF Object Table start at 0.
- Subsequent WMF records MUST refer to an object by its assigned WMF Object Table index. However, there is no requirement that every object defined in the metafile MUST be used.
- An object MUST NOT be used in drawing operations until a META_SELECTOBJECT (section 2.3.4.10) record is received that specifies its WMF Object Table index.
- Later in the processing of the metafile, another META_SELECTOBJECT record might be encountered that selects a different object of the same object type into the current device_context. When that happens, the previously-defined object MUST NOT be deleted, and its index MUST NOT be returned to the pool of available indexes.
- When a META_DELETEOBJECT (section 2.3.4.7) record is received that specifies this object's particular index, the object's resources MUST be released, the binding to its WMF Object Table index MUST be ended, and the index value MUST be returned to the pool of available indexes. The index SHOULD be reused, if needed, by a subsequent object created by another WMF Object record.

Thus, graphics object creation, use, and deletion depend on the correct order of records during playback to achieve the expected results.

Note This document does not mandate that implementations adhere to the implementation of the WMF Object Table presented in this section, as long as the implementation of external behavior is compatible with the behavior described in this document.

3.1.4.2 Object Scaling

A metafile that is created by an application and then passed to another application is likely to be scaled. Scaling may alter the wanted image in a way not anticipated by the creating application that

does not scale the image. Every logical measure defined in a **logical object** is scaled before the object is realized into physical form.

For a logical object such as pens, the width is transformed from logical to physical as an x-scalar value. If the metafile is scaled in y but not in x, the pen width is unchanged. If the metafile is scaled in x but not in y, the pen width does scale. Thus, using a pen of width 1 in a metafile results in a pen that is wider (thick and slow) when the metafile is scaled. If a nominal width pen (width of 1 at all times) is wanted, use 0 as the width because it is not affected by mapping modes. A 0-width pen is drawn as having a width of 1.

Font sizing is more complicated. The two values that scale in a logical font are the height and the width. Most applications use a width of 0 to define a font, which results in a physical font with a width that was designed for the given height. As the metafile is stretched in x, the font remains the same size. As the metafile is stretched in y, however, the physical font grows bigger and probably wider. In and of itself, this is not bad, but problems arise when the metafile makes assumptions about the width of the font by placing the characters of a text string individually by using META_EXTTEXTOUT with a width array or using a META_TEXTOUT for each character. In either case, the x-placement of each character scales with the metafile, but the font's width does not necessarily scale accordingly, which causes characters to overlap or be widely spaced.

The simplest way to overcome this problem is not to place the characters individually but to use META_TEXTOUT (or META_EXTTEXTOUT with no width array) to output the whole string. The text string remains intact, but its size may change in relation to the rest of the image when x and y are not scaled identically. Another possibility is to define the font with a nonzero width so that it scales in x as well as in y. However, doing so is problematic, because its bitmapped fonts may not scale independently in x and y. Scaling a font's width is also possible; unfortunately, any time a font's width is scaled, the look of the typeface changes in ways not necessarily intended by the designers, and a typographically "incorrect" typeface results.

3.1.5 Run-Length Encoding (RLE) Bitmap Compression

Metafile records can contain compressed bitmaps that define their colors with 8 or 4 bits-per-pixel.

Compression forms part of the following member names in the bitmap information header structures for different platforms.

When the **Compression** field of the bitmap information header structure is **BI_RLE8**, an RLE format is used to compress an 8-bit bitmap. This format can be compressed in encoded or absolute modes. Both modes can occur anywhere in the same bitmap:

Encoded mode consists of two bytes. The first byte specifies the number of consecutive pixels to be drawn by using the color index contained in the second byte. In addition, the first byte of the pair can be set to zero to indicate an escape character that denotes the end of a line, the end of a bitmap, or a delta, depending on the value of the second byte. The interpretation of the escape character depends on the value of the second byte of the pair, which can be one of the following values.

Value	Meaning
0	End of line
1	End of bitmap
2	Delta

The 2 bytes following the escape character contain unsigned values indicating the horizontal and vertical offsets of the next pixel from the current position.

In absolute mode, the first byte is zero and the second byte is a value in the range 03H through FFH. The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. When the second byte is two or less, the escape has the same meaning as encoded mode. In absolute mode, each run must be aligned on a word boundary.

The following example shows the hexadecimal values of an 8-bit compressed bitmap:

```
03 04 05 06 00 03 45 56 67 00 02 78 00 02 05 01
02 78 00 00 09 1E 00 01
```

The bitmap expands as follows (two-digit values represent a color index for a single pixel):

```
04 04 04
06 06 06 06 06
45 56 67
78 78
```

Move current position 5 right and 1 down:

```
78 78
end of line
1E 1E 1E 1E 1E 1E 1E 1E 1E
end of RLE bitmap
```

When the **Compression** field is **BI_RLE4**, the bitmap is compressed by using a run-length encoding format for a 4-bit bitmap, which also uses encoded and absolute modes:

In encoded mode, the first byte of the pair contains the number of pixels to be drawn by using the color indexes in the second byte. The second byte contains two color indexes, one in its high-order 4 bits and one in its low-order 4 bits. The first of the pixels is drawn using the color specified by the high-order 4 bits, the second is drawn using the color in the low-order 4 bits, the third is drawn using the color in the high-order 4 bits, and so on, until all the pixels specified by the first byte have been drawn.

In absolute mode, the first byte is zero. The second byte contains the number of color indexes that follow. Subsequent bytes contain color indexes in their high- and low-order 4 bits, one color index for each pixel. In absolute mode, each run must be aligned on a word boundary. The end-of-line, end-of-bitmap, and delta escapes described for **BI_RLE8** also apply to **BI_RLE4** compression.

The following example shows the hexadecimal values of a 4-bit compressed bitmap:

```
03 04 05 06 00 06 45 56 67 00 04 78 00 02 05 01
04 78 00 00 09 1E 00 01
```

The bitmap expands as follows (single-digit values represent a color index for a single pixel):

```
0 4 0
0 6 0 6 0
```

```
4 5 5 6 6 7
7 8 7 8
```

Move current position 5 right and 1 down:

```
7 8 7 8
end of line
1 E 1 E 1 E 1 E 1
end of RLE bitmap
```

3.2 WMF Metafile Example

This section provides an example of a Windows Metafile Format (WMF) metafile, which when processed renders the following image:

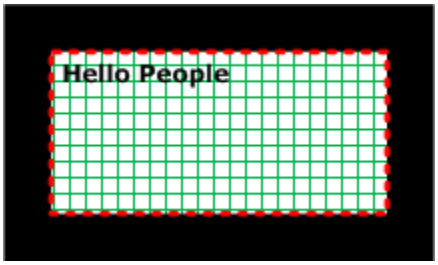


Figure 2: WMF Metafile Example

The contents of this metafile example are shown below in hexadecimal bytes. The far-left column is the byte count; the far-right characters are the interpretation of the bytes in the ANSI Character Set. The sections that follow describe the packets that convey this series of bytes.

```
00000000: 01 00 09 00 00 03 36 00 00 00 02 00 0C 00 00 00 ……..6.........
00000010: 00 00 08 00 00 00 FA 02 04 00 00 00 00 00 00 00 ......ú.........
00000020: FF 00 04 00 00 00 2D 01 00 00 07 00 00 00 FC 02 ÿ.....-.......ü.
00000030: 02 00 00 FF 00 FF 04 00 04 00 00 00 2C 01 01 00 ...ÿ.ÿ......,...
00000040: 07 00 00 00 1B 04 46 00 96 00 00 00 00 00 0C 00 ......F.-.......
00000050: 00 00 21 05 0C 00 48 65 6C 6C 6F 20 50 65 6F 70 ..!...Hello Peop
00000060: 6C 65 0A 00 0A 00 03 00 00 00 00 00             le..........
```

Note When a WMF metafile is processed, the order in which graphics are rendered corresponds to the order of records in the metafile. This may create challenges for devices that have layers. For example, **Printer Command Language (PCL)** defines a graphics layer and a text layer. The text layer in PCL is always drawn on top of the graphics layer. Thus, when converting from metafile format to PCL, the converter, to generate the correct output, MUST either detect text/graphic overlaps and send the text down as graphics, or always send the text as graphics.

The sections that follow provide definitions of the WMF header and records that correspond to this metafile. The WMF Object Table (section 3.1.4.1) refers to an indexed table of WMF Object that are defined in the metafile.

3.2.1 META_HEADER Example

This section provides an example of a Windows Metafile Format (WMF) META_HEADER (section 2.3.2.1) record, which is always the first record in the metafile.

```
00000000: 01 00 09 00 00 03 36 00 00 00 02 00 0C 00 00 00
00000010: 00 00
```

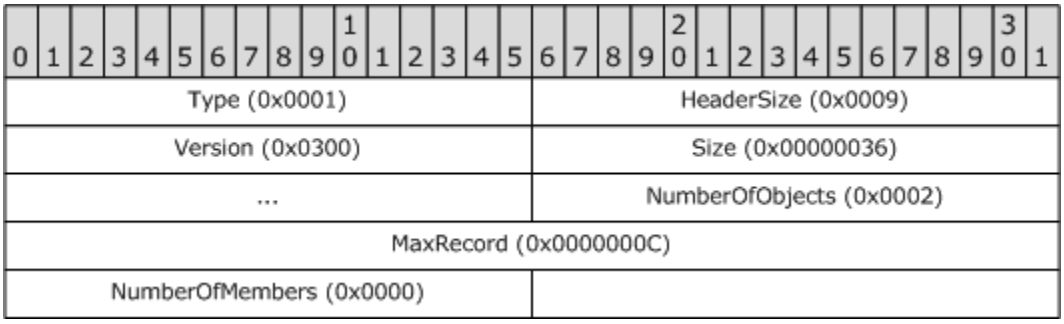


Figure 3: META_HEADER Record Example

Type: 0x0001 specifies the type of metafile from the **MetafileType Enumeration (section 2.1.20)** to be a metafile stored in memory.

HeaderSize: 0x0009 specifies the number of **WORDs** in this record, which is equivalent to 18 (0x0012) bytes.

Version: 0x0300 specifies the metafile version from the **MetafileVersion Enumeration (section 2.1.21)** to be a WMF metafile that supports device-independent bitmaps (DIBs).

Size: 0x00000036 specifies the number of **WORDs** in the entire metafile, which is equivalent to 108 (0x0000006C) bytes.

NumberOfObjects: 0x0002 specifies the number of graphics objects that are defined in the metafile.

MaxRecord: 0x0000000C specifies the size in **WORDs** of the largest record in the metafile, which is equivalent to 24 (0x00000018) bytes.

NumberOfMembers: 0x0000 is not used.

Note Based on the value of the **NumberOfObjects** field, a WMF Object Table (section 3.1.4.1) can be created that is large enough for 2 objects.

3.2.2 META_CREATEPENINDIRECT Example

This section provides an example of a Windows Metafile Format (WMF) META_CREATEPENDIRECT record.

```
00000010:       08 00 00 00 FA 02 04 00 00 00 00 00 00 00
00000020: FF 00
```

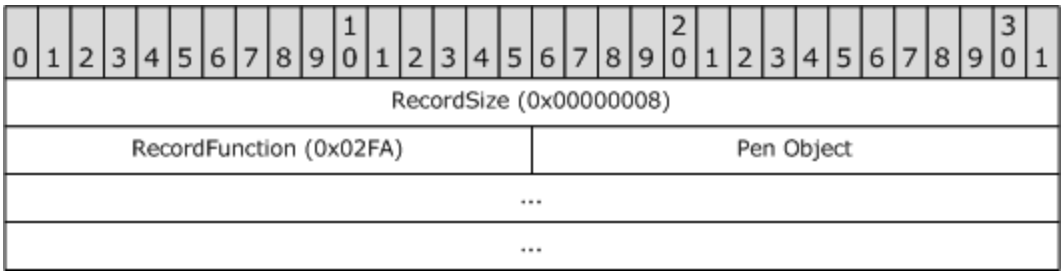


Figure 4: META_CREATEPENINDIRECT Record Example

RecordSize: 0x00000008 specifies the number of **WORDs** in this record, which is equivalent to 16 (0x00000010) bytes.

RecordFunction: 0x02FA specifies a META_CREATEPENINDIRECT record from the **RecordType Enumeration (section 2.1.1)**.

Pen Object: A Pen Object that defines the pen to create.

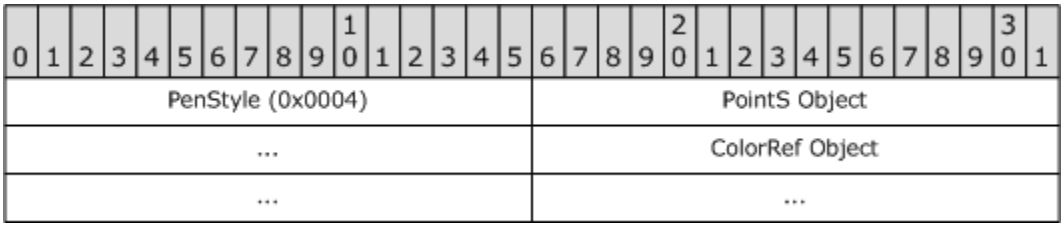


Figure 5: Pen Object Example

PenStyle: 0x0004 specifies the PS_DASHDOTDOT style from the **PenStyle Enumeration (section 2.1.25)**.

PointS Object: A PointS Object that specifies the width of the pen.

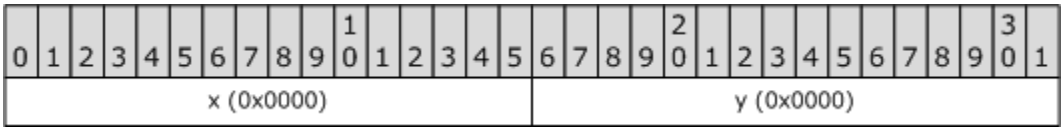


Figure 6: PointS Object Example

x: 0x0000 specifies the width of the pen to be the default, which is 1 pixel.

y: 0x0000 is not used.

ColorRef Object: A ColorRef Object that specifies a green pen color.

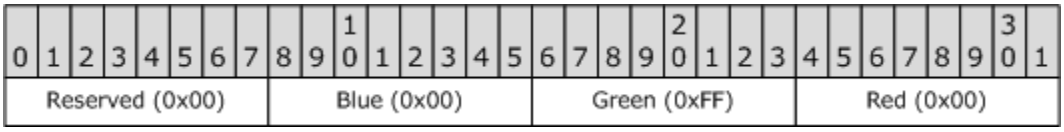


Figure 7: ColorRef Object Example

Reserved: 0x00 is not used.

Blue: 0x00 specifies no blue.

Green: 0xFF specifies full intensity of green.

Red: 0x00 specifies no red.

Note The Pen Object created by processing this record is assigned index 0 in the WMF Object Table (section 3.1.4.1).

3.2.3 META_SELECTOBJECT Example

This section provides an example of a Windows Metafile Format (WMF) META_SELECTOBJECT record.

```
00000020:       04 00 00 00 2D 01 00 00
```

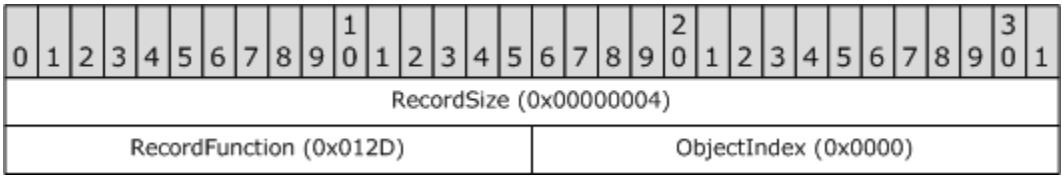


Figure 8: META_SELECTOBJECT Record Example

RecordSize: 0x00000004 specifies the number of **WORDs** in this record, which is equivalent to 8 (0x00000008) bytes.

RecordFunction: 0x012D specifies the type of this record from the **RecordType Enumeration (section 2.1.1)** to be META_SELECTOBJECT.

ObjectIndex: 0x0000 specifies the index in the WMF Object Table (section 3.1.4.1) of the object being selected, which is the Pen Object created in the previous record.

3.2.4 META_CREATEBRUSHINDIRECT Example

This section provides an example of a Windows Metafile Format (WMF) META_CREATEBRUSHINDIRECT record.

```
00000020:                               07 00 00 00 FC 02
00000030: 02 00 00 FF 00 FF 04 00
```

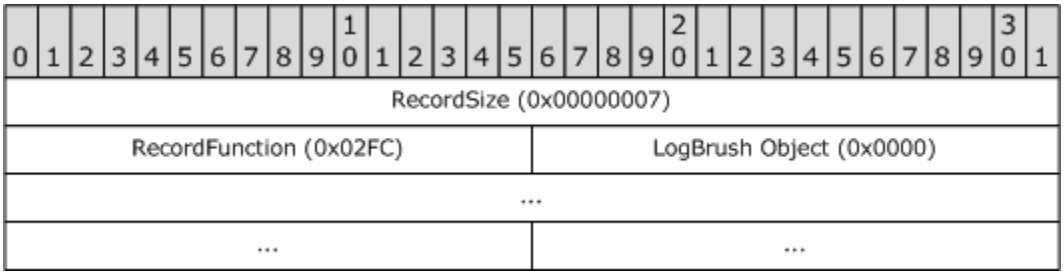


Figure 9: META_CREATEBRUSHINDIRECT Record Example

RecordSize: 0x00000007 specifies the number of **WORDs** in this record, which is equivalent to 14 (0x0000000E) bytes.

RecordFunction: 0x02FC specifies a META_CREATEBRUSHINDIRECT record from the **RecordType Enumeration (section 2.1.1)**.

LogBrush Object: A LogBrush Object that defines the brush to create.

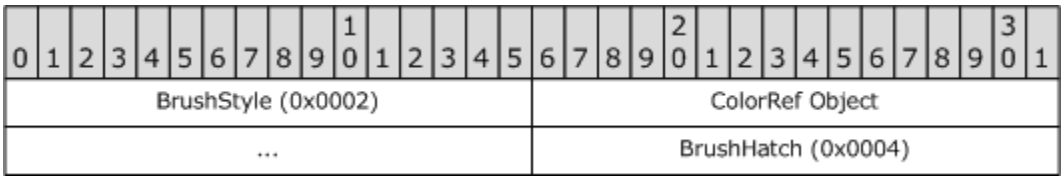


Figure 10: LogBrush Object Example

BrushStyle: 0x0002 specifies the BS_HATCHED style from the **BrushStyle Enumeration (section 2.1.4)**.

ColorRef Object: A ColorRef Object that specifies a magenta brush color.

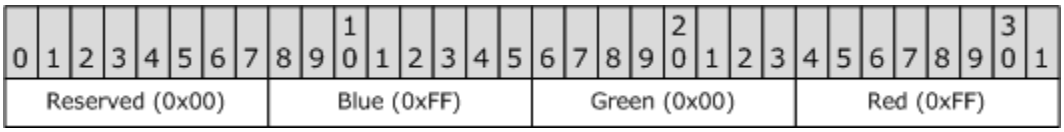


Figure 11: ColorRef Object Example

Reserved: 0x00 is not used.

Blue: 0xFF specifies full intensity blue.

Green: 0x00 specifies no green.

Red: 0xFF specifies full intensity red.

BrushHatch: 0x0004 specifies the hatch style from the **HatchStyle Enumeration (section 2.1.14)**, a horizontal and vertical cross-hatch.

Note The Brush Object created by processing this record is assigned index 1 in the WMF Object Table (section 3.1.4.1).

3.2.5 META_SELECTOBJECT Example

This section provides an example of a Windows Metafile Format (WMF) META_SELECTOBJECT record.

```
00000030:                         04 00 00 00 2D 01 01 00
```

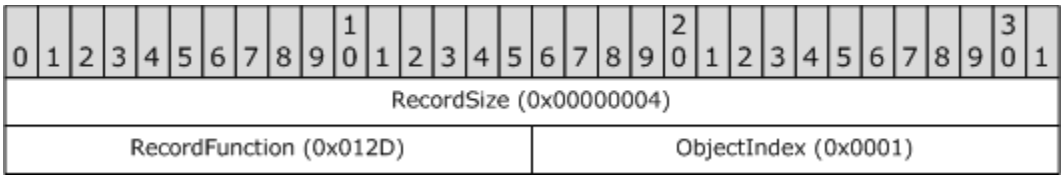


Figure 12: META_SELECTOBJECT Record Example

RecordSize: 0x00000004 specifies the number of **WORDs** in this record, which is equivalent to 8 (0x00000008) bytes.

RecordFunction: 0x012D specifies the type of this record from the **RecordType Enumeration (section 2.1.1)** to be META_SELECTOBJECT.

ObjectIndex: 0x0001 specifies the index in the WMF Object Table (section 3.1.4.1) of the object being selected, which is the Brush Object created in the previous record.

3.2.6 META_RECTANGLE Example

This section provides an example of a Windows Metafile Format (WMF) META_RECTANGLE record.

```
00000040: 07 00 00 00 1B 04 46 00 96 00 00 00 00 00
```

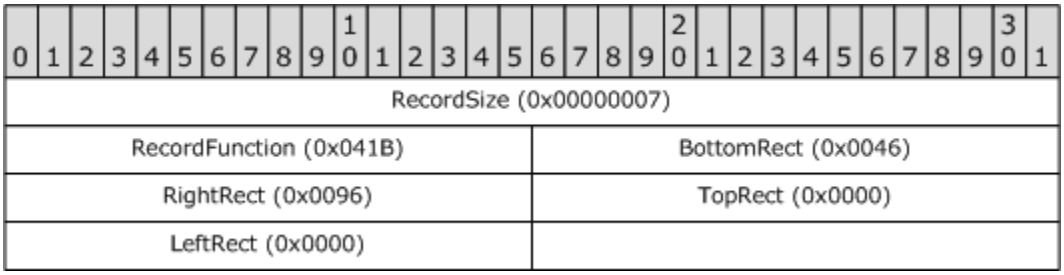


Figure 13: META_RECTANGLE Record Example

RecordSize: 0x0000007 specifies the number of **WORDs** in this record, which is equivalent to 14 (0x0000000E) bytes.

RecordFunction: 0x041B defines the type of this record from the **RecordType Enumeration (section 2.1.1)** to be META_RECTANGLE.

BottomRect: 0x0046 defines the y-coordinate, in logical units, of the lower-right corner of the rectangle.

RightRect: 0x0096 defines the x-coordinate, in logical units, of the lower-right corner of the rectangle.

TopRect: 0x0000 defines the y-coordinate, in logical units, of the upper-left corner of the rectangle.

LeftRect: 0x0000 defines the x-coordinate, in logical units, of the upper-left corner of the rectangle.

3.2.7 META_TEXTOUT Example

This section provides an example of a Windows Metafile Format (WMF) META_TEXTOUT record.

```
0000004E:                                           0C 00
00000050: 00 00 21 05 0C 00 48 65 6C 6C 6F 20 50 65 6F 70
00000060: 6C 65 0A 00 0A 00
```

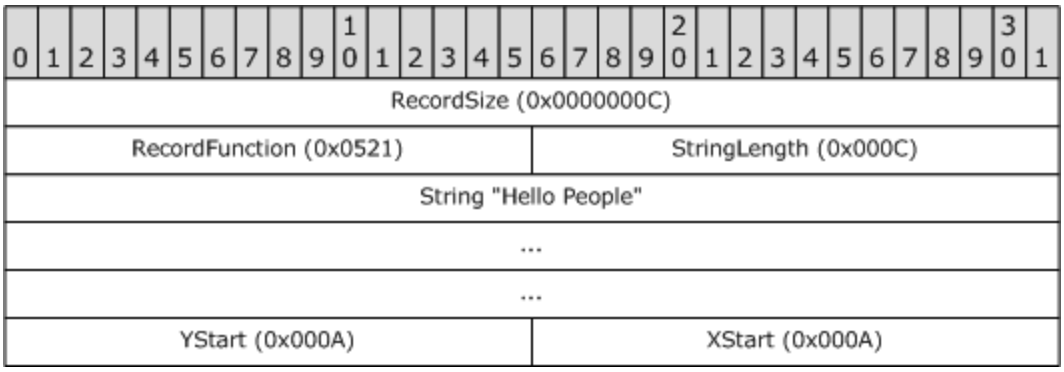


Figure 14: META_TEXTOUT Record Example

RecordSize: 0x0000000C specifies the number of **WORDs** in this record, which is equivalent to 24 (0x00000018) bytes.

RecordFunction: 0x0521 specifies the META_TEXTOUT record from the **RecordType Enumeration (section 2.1.1)**.

StringLength: 0x000C specifies the length of the string in bytes.

String: "Hello People" specifies the text to be drawn.

YStart: 0x000A specifies the vertical (y-axis) coordinate, in logical units, of the point where drawing is to start.

XStart: 0x000A specifies the horizontal (x-axis) coordinate, in logical units, of the point where drawing is to start.

3.2.8 META_EOF Example

This section provides an example of a Windows Metafile Format (WMF) META_EOF record, which is always the last record in the metafile.

```
00000060:                   03 00 00 00 00 00
```

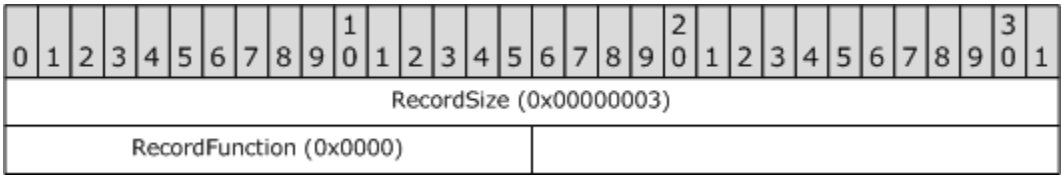


Figure 15: META_EOF Record Example

RecordSize: 0x00000003 specifies the number of **WORDs** in this record, which is equivalent to 6 (0x00000006) bytes.

RecordFunction: 0x0000 specifies the type of this record from the **RecordType Enumeration (section 2.1.1)** to be META_EOF.

4 Security Considerations

This file format enables third parties to send payloads (such as PostScript) to pass through as executable code.

5 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows 95
- Windows 98
- Windows Me
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows NT 3.51
- Windows NT 4.0
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification that is prescribed by using the terms SHOULD or SHOULD NOT implies Windows behavior in accord with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.1: **Image Color Management (ICM)** is the color management technology used on Windows 98, Windows Me, Windows 2000, Windows XP, and Windows Server 2003.

<2> Section 1.1: sRGB is not available on Windows 95, Windows NT 3.1, Windows NT 3.5, or Windows NT 4.0

<3> Section 1.1: **Windows Color System (WCS)** is the color management technology used on Windows Vista and Windows Server 2008.

<4> Section 1.4: Windows NT 3.1, Windows NT 3.51, Windows NT Server 4.0, and Windows 95: The Windows Metafile Format is used primarily with the **Win16** and **Win32s** API's.

<5> Section 2.1.1: For most WMF record types, the high-order byte of the **RecordFunction** field signifies the minimum number of 16-bit parameters, which SHOULD be specified in the record; however, the value is not reliable for that purpose.

Windows does not require that this value is correct and does not use it, with the following exceptions:

- **META_BITBLT**
- **META_DIBBITBLT**
- **META_DIBSTRETCHBLT**
- **META_POLYGON**
- **META_POLYLINE**

- **META_SETPALENTRIES**
- **META_STRETCHBLT**

<6> Section 2.1.4: In Windows implementations, BS_HOLLOW was added as a duplicate symbolic name for BS_NULL, in order for implementers not to confuse it with a null pointer to a brush.

Historically, it was used to satisfy the **Windows Graphics Device Interface (GDI)** requirement for a brush, in situations when the GDI application did not require any kind of brush to be used.

<7> Section 2.1.5: Not supported on Windows NT 3.1 or Windows NT 3.51.

<8> Section 2.1.6: Windows Vista and Windows Server 2008: This value is ignored.

<9> Section 2.1.6: Windows Vista and Windows Server 2008: This value is ignored.

<10> Section 2.1.6: Windows Vista and Windows Server 2008: This value is ignored.

<11> Section 2.1.6: Windows 2000, Windows XP, and Windows Server 2003: Font association is turned off.

Windows Vista and Windows Server 2008: This value is ignored.

Windows XP SP1: Turns off font association for the font. Note that this value is not guaranteed to have any effect on any platform after Windows Server 2003.

<12> Section 2.1.19: Windows Vista and Windows Server 2008: This function is not supported.

<13> Section 2.1.30: In Windows implementations, the PostScript printer driver uses a default line join style.

<14> Section 2.1.31: Windows 95 and Windows 98: The symbolic name "STRETCH_ANDSCANS" is synonymous with this value.

<15> Section 2.1.31: Windows 95 and Windows 98: The symbolic name "STRETCH_ORSCANS" is synonymous with this value.

<16> Section 2.1.31: Windows 95 and Windows 98: The symbolic name "STRETCH_DELETESCANS" is synonymous with this value.

<17> Section 2.1.31: Windows 95 and Windows 98: The symbolic name "STRETCH_HALFTONE" is synonymous with this value.

<18> Section 2.2.1.3: Windows 95, Windows NT 4.0, Windows 98, Windows Me, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 support this structure.

<19> Section 2.2.1.4: Windows 98, Windows Me, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 support this structure.

<20> Section 2.2.1.8: In Windows implementations, BS_HOLLOW was added as a duplicate symbolic name for BS_NULL, in order for implementers not to confuse it with a null pointer to a brush.

Historically, it was used to satisfy the **Windows Graphics Device Interface (GDI)** requirement for a brush in situations where the **GDI** application did not require any kind of brush to be used.

<21> Section 2.2.1.8: Windows Vista and Windows Server 2008 create a solid-color black brush by default, when a **BrushHatch** value of BS_PATTERN or BS_DIBPATTERNPT is present. The behavior on other versions of Windows is undefined.

<22> Section 2.2.2.4: All Windows versions: Mapping the logical font size to the available physical fonts occurs the first time the logical font needs to be used in a drawing operation.

For the MM_TEXT mapping mode, the following formula can be used to compute the height of a font with a specified point size:

```
Height = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

<23> Section 2.2.2.9: Windows sets this field to 0x0000.

<24> Section 2.2.2.9: Windows sets this field to an arbitrary value.

<25> Section 2.3.4.1: Windows Vista and Windows Server 2008 create a solid-color black brush by default, when a **BrushHatch** value of BS_PATTERN or BS_DIBPATTERNPT is present. The behavior on other versions of Windows is undefined.

6 Index

N

O

P

Q

R

S

T

V

W