[MS-EMFPLUS]:

Enhanced Metafile Format Plus Extensions

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
2/22/2007	0.01	New	Version 0.01 release
6/1/2007	1.0	Major	Updated and revised the technical content.
7/3/2007	1.1	Minor	Editorial changes to make consistent references and use of acronyms; and some glossary terms were added.
7/20/2007	1.2	Minor	Clarified the meaning of the technical content.
8/10/2007	1.3	Minor	Clarified the meaning of the technical content.
9/28/2007	1.4	Minor	Clarified the meaning of the technical content.
10/23/2007	2.0	Major	Updated and revised the technical content; restructured for easier navigation.
11/30/2007	3.0	Major	Restructured EMF+ constants, objects, other revisions, and standardized art
1/25/2008	3.1	Minor	Clarified the meaning of the technical content.
3/14/2008	4.0	Major	Abstract data model added.
5/16/2008	4.0.1	Editorial	Changed language and formatting in the technical content.
6/20/2008	4.1	Minor	Clarified the meaning of the technical content.
7/25/2008	4.1.1	Editorial	Changed language and formatting in the technical content.
8/29/2008	4.1.2	Editorial	Changed language and formatting in the technical content.
10/24/2008	4.2	Minor	Clarified the meaning of the technical content.
12/5/2008	4.3	Minor	Clarified the meaning of the technical content.
1/16/2009	5.0	Major	Updated and revised the technical content.
2/27/2009	5.1	Minor	Clarified the meaning of the technical content.
4/10/2009	6.0	Major	Updated and revised the technical content.
5/22/2009	7.0	Major	Updated and revised the technical content.
7/2/2009	7.1	Minor	Clarified the meaning of the technical content.
8/14/2009	7.1.1	Editorial	Changed language and formatting in the technical content.
9/25/2009	7.2	Minor	Clarified the meaning of the technical content.
11/6/2009	7.3	Minor	Clarified the meaning of the technical content.
12/18/2009	7.3.1	Editorial	Changed language and formatting in the technical content.
1/29/2010	7.4	Minor	Clarified the meaning of the technical content.
3/12/2010	7.5	Minor	Clarified the meaning of the technical content.
4/23/2010	7.5.1	Editorial	Changed language and formatting in the technical content.
6/4/2010	7.6	Minor	Clarified the meaning of the technical content.

Date	Revision History	Revision Class	Comments
7/16/2010	8.0	Major	Updated and revised the technical content.
8/27/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2010	8.1	Minor	Clarified the meaning of the technical content.
11/19/2010	8.2	Minor	Clarified the meaning of the technical content.
1/7/2011	8.2	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	8.2	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	8.2	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	8.2	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	8.3	Minor	Clarified the meaning of the technical content.
9/23/2011	9.0	Major	Updated and revised the technical content.
12/16/2011	10.0	Major	Updated and revised the technical content.
3/30/2012	10.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	10.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	10.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	10.0	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	11.0	Major	Updated and revised the technical content.
11/14/2013	12.0	Major	Updated and revised the technical content.
2/13/2014	12.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	13.0	Major	Updated and revised the technical content.
6/30/2015	14.0	Major	Significantly changed the technical content.
10/16/2015	14.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	14.1	Minor	Clarified the meaning of the technical content.

Table of Contents

1 Introduction 10
1.1 Glossary 10
1.2 References 17
1.2.1 Normative References 17
1.2.2 Informative References 18
1.3 Overview 18
1.3.1 Metafile Structure 18
1.3.2 Byte Ordering 20
1.4 Relationship to Protocols and Other Structures 21
1.5 Applicability Statement 21
1.6 Versioning and Localization 21
1.7 Vendor-Extensible Fields 22

2 Structures 23
2.1 EMF+ Constants 23
2.1.1 Enumeration Constant Types 23
2.1.1.1 RecordType Enumeration 25
2.1.1.2 BitmapDataType Enumeration 28
2.1.1.3 BrushType Enumeration 29
2.1.1.4 CombineMode Enumeration 29
2.1.1.5 CompositingMode Enumeration 30
2.1.1.6 CompositingQuality Enumeration 30
2.1.1.7 CurveAdjustments Enumeration 31
2.1.1.8 CurveChannel Enumeration 32
2.1.1.9 CustomLineCapDataType Enumeration 32
2.1.1.10 DashedLineCapType Enumeration 32
2.1.1.11 FilterType Enumeration 33
2.1.1.12 GraphicsVersion Enumeration 33
2.1.1.13 HatchStyle Enumeration 34
2.1.1.14 HotkeyPrefix Enumeration 37
2.1.1.15 ImageDataType Enumeration 38
2.1.1.16 InterpolationMode Enumeration 38
2.1.1.17 LanguageIdentifier Enumeration 39
2.1.1.18 LineCapType Enumeration 50
2.1.1.19 LineJoinType Enumeration 51
2.1.1.20 LineStyle Enumeration 51
2.1.1.21 MetafileDataType Enumeration 52
2.1.1.22 ObjectType Enumeration 52
2.1.1.23 PathPointType Enumeration 53
2.1.1.24 PenAlignment Enumeration 53
2.1.1.25 PixelFormat Enumeration 54
2.1.1.26 PixelOffsetMode Enumeration 55
2.1.1.27 RegionNodeDataType Enumeration 56
2.1.1.28 SmoothingMode Enumeration 57
2.1.1.29 StringAlignment Enumeration 57
2.1.1.30 StringDigitSubstitution Enumeration 58
2.1.1.31 StringTrimming Enumeration 58
2.1.1.32 TextRenderingHint Enumeration 59
2.1.1.33 UnitType Enumeration 59
2.1.1.34 WrapMode Enumeration 60
2.1.2 Bit Flag Constant Types 60
2.1.2.1 BrushData Flags 61
2.1.2.2 CustomLineCapData Flags 62
2.1.2.3 DriverStringOptions Flags 62
2.1.2.4 FontStyle Flags 62

2.1.2.5 PaletteStyle Flags 63
2.1.2.6 PathPointType Flags 63
2.1.2.7 PenData Flags 64
2.1.2.8 StringFormat Flags 64
2.1.3 Standard Identifier Constant Types 66
2.1.3.1 ImageEffects Identifiers 66
2.2 EMF+ Objects 67
2.2.1 Graphics Object Types 67
2.2.1.1 EmfPlusBrush Object 67
2.2.1.2 EmfPlusCustomLineCap Object 68
2.2.1.3 EmfPlusFont Object 69
2.2.1.4 EmfPlusImage Object 70
2.2.1.5 EmfPlusImageAttributes Object 70
2.2.1.6 EmfPlusPath Object 71
2.2.1.7 EmfPlusPen Object 72
2.2.1.8 EmfPlusRegion Object 73
2.2.1.9 EmfPlusStringFormat Object 74
2.2.2 Structure Object Types 75
2.2.2.1 EmfPlusARGB Object 77
2.2.2.2 EmfPlusBitmap Object 78
2.2.2.3 EmfPlusBitmapData Object 79
2.2.2.4 EmfPlusBlendColors Object 80
2.2.2.5 EmfPlusBlendFactors Object 81
2.2.2.6 EmfPlusBoundaryPathData Object 82
2.2.2.7 EmfPlusBoundaryPointData Object 82
2.2.2.8 EmfPlusCharacterRange Object 83
2.2.2.9 EmfPlusCompoundLineData Object 83
2.2.2.10 EmfPlusCompressedImage Object 84
2.2.2.11 EmfPlusCustomEndCapData Object 84
2.2.2.12 EmfPlusCustomLineCapArrowData Object 85
2.2.2.13 EmfPlusCustomLineCapData Object 86
2.2.2.14 EmfPlusCustomLineCapOptionalData Object 87
2.2.2.15 EmfPlusCustomStartCapData Object 88
2.2.2.16 EmfPlusDashedLineData Object 88
2.2.2.17 EmfPlusFillPath Object 89
2.2.2.18 EmfPlusFocusScaleData Object 89
2.2.2.19 EmfPlusGraphicsVersion Object 90
2.2.2.20 EmfPlusHatchBrushData Object 90
2.2.2.21 EmfPlusInteger7 Object 91
2.2.2.22 EmfPlusInteger15 Object 91
2.2.2.23 EmfPlusLanguageIdentifier Object 92
2.2.2.24 EmfPlusLinearGradientBrushData Object 92
2.2.2.25 EmfPlusLinearGradientBrushOptionalData Object 93
2.2.2.26 EmfPlusLinePath Object 94
2.2.2.27 EmfPlusMetafile Object 95
2.2.2.28 EmfPlusPalette Object 95
2.2.2.29 EmfPlusPathGradientBrushData Object 96
2.2.2.30 EmfPlusPathGradientBrushOptionalData Object 98
2.2.2.31 EmfPlusPathPointType Object 99
2.2.2.32 EmfPlusPathPointTypeRLE Object 99
2.2.2.33 EmfPlusPenData Object 99
2.2.2.34 EmfPlusPenOptionalData Object 100
2.2.2.35 EmfPlusPoint Object 102
2.2.2.36 EmfPlusPointF Object 103
2.2.2.37 EmfPlusPointR Object 103
2.2.2.38 EmfPlusRect Object 103
2.2.2.39 EmfPlusRectF Object 104
2.2.2.40 EmfPlusRegionNode Object 104

2.2.2.41 EmfPlusRegionNodeChildNodes Object 105
2.2.2.42 EmfPlusRegionNodePath Object 105
2.2.2.43 EmfPlusSolidBrushData Object 106
2.2.2.44 EmfPlusStringFormatData Object 106
2.2.2.45 EmfPlusTextureBrushData Object 107
2.2.2.46 EmfPlusTextureBrushOptionalData Object 107
2.2.2.47 EmfPlusTransformMatrix Object 108
2.2.3 Image Effects Object Types 109
2.2.3.1 BlurEffect Object 109
2.2.3.2 BrightnessContrastEffect Object 110
2.2.3.3 ColorBalanceEffect Object 111
2.2.3.4 ColorCurveEffect Object 112
2.2.3.5 ColorLookupTableEffect Object 113
2.2.3.6 ColorMatrixEffect Object 114
2.2.3.7 HueSaturationLightnessEffect Object 117
2.2.3.8 LevelsEffect Object 118
2.2.3.9 RedEyeCorrectionEffect Object 119
2.2.3.10 SharpenEffect Object 120
2.2.3.11 TintEffect Object 120
2.3 EMF+ Records 121
2.3.1 Clipping Record Types 121
2.3.1.1 EmfPlusOffsetClip Record 122
2.3.1.2 EmfPlusResetClip Record 123
2.3.1.3 EmfPlusSetClipPath Record 123
2.3.1.4 EmfPlusSetClipRect Record 124
2.3.1.5 EmfPlusSetClipRegion Record 125
2.3.2 Comment Record Types 126
2.3.2.1 EmfPlusComment Record 126
2.3.3 Control Record Types 127
2.3.3.1 EmfPlusEndOfFile Record 128
2.3.3.2 EmfPlusGetDC Record 128
2.3.3.3 EmfPlusHeader Record 129
2.3.4 Drawing Record Types 130
2.3.4.1 EmfPlusClear Record 132
2.3.4.2 EmfPlusDrawArc Record 133
2.3.4.3 EmfPlusDrawBeziers Record 134
2.3.4.4 EmfPlusDrawClosedCurve Record 136
2.3.4.5 EmfPlusDrawCurve Record 139
2.3.4.6 EmfPlusDrawDriverString Record 141
2.3.4.7 EmfPlusDrawEllipse Record 143
2.3.4.8 EmfPlusDrawImage Record 144
2.3.4.9 EmfPlusDrawImagePoints Record 146
2.3.4.10 EmfPlusDrawLines Record 148
2.3.4.11 EmfPlusDrawPath Record 150
2.3.4.12 EmfPlusDrawPie Record 151
2.3.4.13 EmfPlusDrawRects Record 152
2.3.4.14 EmfPlusDrawString Record 153
2.3.4.15 EmfPlusFillClosedCurve Record 155
2.3.4.16 EmfPlusFillEllipse Record 158
2.3.4.17 EmfPlusFillPath Record 159
2.3.4.18 EmfPlusFillPie Record 160
2.3.4.19 EmfPlusFillPolygon Record 162
2.3.4.20 EmfPlusFillRects Record 164
2.3.4.21 EmfPlusFillRegion Record 165
2.3.5 Object Record Types 166
2.3.5.1 EmfPlusObject Record 167
2.3.5.2 EmfPlusSerializableObject Record 168
2.3.6 Property Record Types 169

2.3.6.1 EmfPlusSetAntiAliasMode Record 171
2.3.6.2 EmfPlusSetCompositingMode Record 171
2.3.6.3 EmfPlusSetCompositingQuality Record 172
2.3.6.4 EmfPlusSetInterpolationMode Record 173
2.3.6.5 EmfPlusSetPixelOffsetMode Record 173
2.3.6.6 EmfPlusSetRenderingOrigin Record 174
2.3.6.7 EmfPlusSetTextContrast Record 175
2.3.6.8 EmfPlusSetTextRenderingHint Record 176
2.3.7 State Record Types 176
2.3.7.1 EmfPlusBeginContainer Record 177
2.3.7.2 EmfPlusBeginContainerNoParams Record 179
2.3.7.3 EmfPlusEndContainer Record 179
2.3.7.4 EmfPlusRestore Record 180
2.3.7.5 EmfPlusSave Record 181
2.3.8 Terminal Server Record Types 181
2.3.8.1 EmfPlusSetTSClip Record 182
2.3.8.2 EmfPlusSetTSGraphics Record 184
2.3.9 Transform Record Types 185
2.3.9.1 EmfPlusMultiplyWorldTransform Record 187
2.3.9.2 EmfPlusResetWorldTransform Record 187
2.3.9.3 EmfPlusRotateWorldTransform Record 188
2.3.9.4 EmfPlusScaleWorldTransform Record 189
2.3.9.5 EmfPlusSetPageTransform Record 190
2.3.9.6 EmfPlusSetWorldTransform Record 191
2.3.9.7 EmfPlusTranslateWorldTransform Record 191

3 Structure Examples 193
3.1 Metafile Design 193
3.1.1 Byte Ordering Example 193
3.1.2 Managing Graphics Objects 193
3.2 EMF+ Metafile Example 194
3.2.1 EMR_HEADER Example 197
3.2.2 EMR_COMMENT_EMFPLUS Example 1 199
3.2.2.1 EmfPlusHeader Example 200
3.2.3 EMR_SAVEDC Example 1 200
3.2.4 EMR_SETICMMODE Example 1 201
3.2.5 EMR_SAVEDC Example 2 201
3.2.6 EMR_SETICMMODE Example 2 202
3.2.7 EMR_SAVEDC Example 3 202
3.2.8 EMR_SETLAYOUT Example 1 202
3.2.9 EMR_SETMETARGN Example 1 203
3.2.10 EMR_SELECTOBJECT Example 1 203
3.2.11 EMR_SELECTOBJECT Example 2 204
3.2.12 EMR_SELECTOBJECT Example 3 204
3.2.13 EMR_SELECTPALETTE Example 1 204
3.2.14 EMR_SETBKCOLOR Example 1 205
3.2.15 EMR_SETTEXTCOLOR Example 1 205
3.2.16 EMR_SETBKMODE Example 1 206
3.2.17 EMR_SETPOLYFILLMODE Example 1 206
3.2.18 EMR_SETROP2 Example 1 207
3.2.19 EMR_SETSTRETCHBLTMODE Example 1 207
3.2.20 EMR_SETTEXTALIGN Example 1 207
3.2.21 EMR_SETBRUSHORGEX Example 1 208
3.2.22 EMR_SETMITERLIMIT Example 1 208
3.2.23 EMR_MOVETOEX Example 1 209
3.2.24 EMR_SETWORLDTRANSFORM Example 1 209
3.2.25 EMR_MODIFYWORLDTRANSFORM Example 1 210
3.2.26 EMR_SETLAYOUT Example 2 211

3.2.27 EMR_SETBRUSHORGEX Example 2 211
3.2.28 EMR_EXTCREATEFONTINDIRECTW Example 212
3.2.29 EMR_SELECTOBJECT Example 4 214
3.2.30 EMR_SELECTOBJECT Example 5 215
3.2.31 EMR_DELETEOBJECT Example 215
3.2.32 EMR_COMMENT_EMFPLUS Example 2 215
3.2.32.1 EmfPlusSetAntiAliasMode Example 1 216
3.2.32.2 EmfPlusSetCompositingQuality Example 1 216
3.2.32.3 EmfPlusSetInterpolationMode Example 1 217
3.2.32.4 EmfPlusSetPixelOffsetMode Example 1 217
3.2.32.5 EmfPlusSetTextRenderingHint Example 1 218
3.2.32.6 EmfPlusMultiplyWorldTransform Example 1 218
3.2.32.7 EmfPlusSave Example 219
3.2.32.8 EmfPlusMultiplyWorldTransform Example 2 220
3.2.32.9 EmfPlusSetWorldTransform Example 1 220
3.2.32.10 EmfPlusBeginContainerNoParams Example 221
3.2.32.11 EmfPlusSetAntiAliasMode Example 2 222
3.2.32.12 EmfPlusSetCompositingQuality Example 2 222
3.2.32.13 EmfPlusSetInterpolationMode Example 2 223
3.2.32.14 EmfPlusSetPixelOffsetMode Example 2 223
3.2.32.15 EmfPlusSetTextRenderingHint Example 2 223
3.2.32.16 EmfPlusSetPageTransform Example 224
3.2.32.17 EmfPlusSetWorldTransform Example 2 224
3.2.32.18 EmfPlusSetWorldTransform Example 3 225
3.2.32.19 EmfPlusSetWorldTransform Example 4 226
3.2.32.20 EmfPlusSetWorldTransform Example 5 227
3.2.32.21 EmfPlusObject Example 227
3.2.32.22 EmfPlusFillPath Example 229
3.2.33 EMR_COMMENT_EMFPLUS Example 3 229
3.2.33.1 EmfPlusObject Example 230
3.2.33.2 EmfPlusDrawPath Example 232
3.2.34 EMR_RESTOREDC Example 1 232
3.2.35 EMR_RESTOREDC Example 2 233
3.2.36 EMR_SELECTOBJECT Example 6 233
3.2.37 EMR_RESTOREDC Example 3 233
3.2.38 EMR_SAVEDC Example 4 234
3.2.39 EMR_SETLAYOUT Example 3 234
3.2.40 EMR_SETMETARGN Example 2 235
3.2.41 EMR_SELECTOBJECT Example 7 235
3.2.42 EMR_SELECTOBJECT Example 8 235
3.2.43 EMR_SELECTOBJECT Example 9 236
3.2.44 EMR_SELECTPALETTE Example 2 236
3.2.45 EMR_SETBKCOLOR Example 2 237
3.2.46 EMR_SETTEXTCOLOR Example 2 237
3.2.47 EMR_SETBKMODE Example 2 237
3.2.48 EMR_SETPOLYFILLMODE Example 2 238
3.2.49 EMR_SETROP2 Example 2 238
3.2.50 EMR_SETSTRETCHBLTMODE Example 2 239
3.2.51 EMR_SETTEXTALIGN Example 2 239
3.2.52 EMR_SETBRUSHORGEX Example 3 240
3.2.53 EMR_SETMITERLIMIT Example 2 240
3.2.54 EMR_MOVETOEX Example 2 241
3.2.55 EMR_SETWORLDTRANSFORM Example 2 241
3.2.56 EMR_MODIFYWORLDTRANSFORM Example 2 242
3.2.57 EMR_SETLAYOUT Example 4 242
3.2.58 EMR_COMMENT_BEGINGROUP Example 243
3.2.59 EMR_SETWORLDTRANSFORM Example 3 244
3.2.60 EMR_MODIFYWORLDTRANSFORM Example 3 244

3.2.61 EMR_MODIFYWORLDTRANSFORM Example 4 245
3.2.62 EMR_MODIFYWORLDTRANSFORM Example 5 246
3.2.63 EMR_SELECTOBJECT Example 10 246
3.2.64 EMR_ROUNDRECT Example....247
3.2.65 EMR_COMMENT_ENDGROUP Example 247
3.2.66 EMR_RESTOREDC Example 4 248
3.2.67 EMR_COMMENT_EMFPLUS Example 4....248
3.2.67.1 EmfPlusEndContainer Example....249
3.2.67.2 EmfPlusRestore Example....249
3.2.67.3 EmfPlusSetWorldTransform Example 250
3.2.68 EMR_BITBLT Example 251
3.2.69 EMR_RESTOREDC Example 5 252
3.2.70 EMR_COMMENT_EMFPLUS Example 5....253
3.2.70.1 EmfPlusEndOfFile Example 253
3.2.71 EMR_EOF Example....254
3.3 EMF+ String Drawing Example....254
4 Security....256
5 Appendix A: Product Behavior....257
6 Change Tracking....260
7 Index....262

1 Introduction

This is a specification of the Enhanced Metafile Format Plus Extensions (EMF+) structure. The **EMF+** structure specifies a **metafile** format that can store a picture in device-independent form. The stored picture can be rendered by parsing and processing the metafile.

An EMF+ metafile is a series of variable-length records, called EMF+ records, which contain graphics drawing commands, object definitions, and properties. The metafile begins with a header record, which includes the metafile version, its size, the resolution of the device on which the picture was created, and the dimensions of the picture. An EMF+ metafile is "played back" when its records are converted to a format understood by a specific graphics device. An image defined in an EMF+ structure maintains its dimensions, shape, and proportions on any output device, including printers, plotters, and desktops, or in the client areas of applications.

Sections 1.7 and 2 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

affine transform: A matrix operation that consists of a linear **transform** followed by a **translation**. An affine transform can be used to correct perspective distortions by transforming the measurements from **world space** to **device space** coordinates.

alpha blending: In computer graphics, the process of combining an image with a background to create the appearance of partial transparency. The extent of blending is determined by the value of the **alpha** component of the color being rendered.

alpha transparency: An alpha value is a transparency value represented by a number between zero and one. Each pixel has an alpha value that represents its level of transparency, which is multiplied by the color values to get the final value. Each pixel has an alpha value that represents its level of transparency.

American National Standards Institute (ANSI) character set: A character set defined by a code page approved by the American National Standards Institute (ANSI). The term "ANSI" as used to signify Windows code pages is a historical reference and a misnomer that persists in the Windows community. The source of this misnomer stems from the fact that the Windows code page 1252 was originally based on an ANSI draft, which became International Organization for Standardization (ISO) Standard 8859-1 [ISO/IEC-8859-1]. In Windows, the ANSI character set can be any of the following code pages: 1252, 1250, 1251, 1253, 1254, 1255, 1256, 1257, 1258, 874, 932, 936, 949, or 950. For example, "ANSI application" is usually a reference to a non-**Unicode** or code-page-based application. Therefore, "ANSI character set" is often misused to refer to one of the character sets defined by a Windows code page that can be used as an active system code page; for example, character sets defined by code page 1252 or character sets defined by code page 950. Windows is now based on **Unicode**, so the use of ANSI character sets is strongly discouraged unless they are used to interoperate with legacy applications or legacy data.

anti-aliasing: The smoothing of the jagged appearance of font characters and lines, which is an artifact of the limited resolution on an output device. The pixels that surround the edges of the character glyph or line are changed to varying shades of color in order to blend the sharp edge into the background.

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit

ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

baseline: The imaginary line to which the bottom of the lowercase "x" character in a font **typeface** is aligned.

Bezier curve: A type of curve, defined by a mathematical formula and a number of points greater than or equal to two, which is used in computer graphics and in the mathematical field of numeric analysis. A cubic **Bezier curve** is defined by four points: two endpoints and two control points. The curve does not pass through the control points, but the control points act like magnets, pulling the curve in certain directions and influencing the way the curve bends. With multiple **Bezier curves**, the endpoint of one is the starting point of the next.

big-endian: Multiple-byte values that are byte-ordered with the most significant byte stored in the memory location with the lowest address.

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

black saturation: The low end of the range of **saturation** for a **color channel**.

blur effect: An **image effect** that is used to remove detail, resulting in an image that appears as if viewed through a translucent lens. A Gaussian blur uses a Gaussian distribution to calculate changes to individual pixels in the image. A Gaussian distribution is a statistical probability distribution that produces a "bell curve".

box filter: An **anti-aliasing** algorithm that averages the pixels in a rectangular area to compute a new value for the pixels. It is useful only for reducing the size of images.

brightness: The relative lightness or darkness of an image, or of a particular color in an image.

brightness contrast effect: An **image effect** that is used to change the **intensity** of an image by expanding or contracting the lightest and darkest areas of an image.

cardinal spline: A **spline** with curves that pass smoothly through each of its control points. The degree of curvature is defined by a tension parameter. The curve has no sharp corners or abrupt changes. A complete derivation of the cubic polynomials for the canonical **spline** can be found in [SPLINE77]. See also [PETZOLD] for more information.

cell height: A vertical measure of font size, which is the sum of the font height and internal leading. It might not be the same as the distance between two lines of text.

ClearType: A font technology developed by Microsoft that can display fractions of pixels of character glyphs and which improves the readability of text on liquid crystal displays (LCDs) and flat-panel monitors.

closed cardinal spline: A **cardinal spline** with a curve that passes through the last defined point and connects with the first.

color balance effect: An **image effect** that produces optimal color display by adjusting the relative amounts of red, green, and blue in the image.

color channel: A component color from which all colors in an image are rendered. In an RGB color space, there are color channels for red, green, and blue. In a grayscale color space, the color channels are black and white. Color channel values typically range from 0 to 255.

color curve: A graphical mechanism for displaying and adjusting color measurements of an image. The points on a color curve correspond to the pure colors in a color space (1).

color curve effect: An **image effect** that is used to apply one of eight adjustments to the **color curve** of an image: **exposure**, **density**, **contrast**, **highlight**, **shadow**, **midtone**, **white saturation**, and **black saturation**.

color lookup table effect: An **image effect** that is used to make custom color adjustments to images. A lookup table is defined for four individual **color channels**: **alpha** (transparency), red, green, and blue. Each lookup table is an array of 256 bytes that can be set to specific values.

color mapping: The process of associating integer color indices with **color channel** values.

color matrix: A matrix of floating-point values from zero to one, inclusive, that can be multiplied with a **color vector** to effect a color **transform**. A 4x4 matrix can be used to perform linear **transforms**, and a 5x5 matrix can be used to perform nonlinear **transforms**.

color matrix effect: An **image effect** that uses a 5x5 **color matrix** to perform an **affine transform** to the **color vectors** of an image.

color vector: An **RGB** plus **alpha** value that represents a specific color and transparency. Each value is in the range of zero to one, inclusive; for red, green and blue, zero means no **intensity** of the color and one means maximum **intensity**.

color wheel: An organization of color **hues** around a circle, showing relationships between colors considered to be primary, secondary, and complementary. In an **RGB** color space, red, green, and blue primary colors are arranged at equally spaced points around the circle. Magenta, yellow, and cyan secondary colors and tertiary mixtures are located at intermediate points on the circle. The center is white or gray.

compositing: The process that takes place during image rendering, which combines color data from multiple graphics region.

contrast: The relative difference between lightness and darkness in an area of an image.

coordinate space: A space based on Cartesian coordinates, which provides a means of specifying the location of each point in the space. A two-dimensional coordinate space requires two axes that are perpendicular and equal in length. Three two-dimensional coordinate spaces are generally used to describe an output surface: world, page, and device. To scale device-independent output for a particular physical device, a rectangular area in the world or page coordinate space is mapped into the device coordinate space using a **transform**

density: A measure of image opacity; that is, the amount of light that passes through photographic film at a particular location.

device context: A collection of properties and objects that defines a dynamic environment for processes on a device. For graphics output, properties include brush style, line style, text layout, foreground and background colors, and mapping mode; and objects include a brush, pen, font, **palette**, **region**, and transform matrix. Multiple device contexts can exist simultaneously, but a single device context specifies the environment for graphics output at a particular point in time.

device driver: The software that the system uses to communicate with a device such as a display, printer, mouse, or communications adapter. An abstraction layer that restricts access of applications to various hardware devices on a given computer system. It is often referred to simply as a "driver".

device space: The output space for graphics **transforms**. It usually refers to the client area of an application window; however, it can also include the entire desktop, a complete window, or a page of printer or plotter paper. Physical device space dimensions vary according to the dimensions set by the display, printer, or plotter technology.

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**.

em size: A measure of font size, which is the **cell height** minus the internal leading. An "em" is a term that has been used historically as a unit of typeset size.

enhanced metafile format (EMF): A file format that supports the device-independent definitions of images.

enhanced metafile format plus extensions (EMF+): A file format that supports the device-independent definitions of images.

enhanced metafile spool format (EMFSPOOL): A format that specifies a structure of **enhanced metafile format (EMF)** records used for defining application and device-independent printer spool files.

Exchangeable Image File Format (EXIF): A de facto standard format for storing files containing digital photographic images and audio files. EXIF uses existing formats for data compression, including **JPEG** and TIFF; but it is not supported by older versions of **JPEG**, **PNG**, or **GIF**. EXIF specifies metadata tags for storing information about a photographic image, including camera make and model, shutter speed, exposure compensation, F-stop, the metering system, whether a flash was used, the date and time the photograph was taken, auxiliary lenses that were used, the resolution, and a thumbnail image for previewing the photograph. EXIF is specified in [EXIF].

exposure: A measure of the amount of light in which a photographic image is recorded. Overexposed images are lighter than normal; underexposed images are darker than normal.

font axis: A property of font design that can assume a linear range of values. In general, a font has multiple axes. For example, a font may define an axis for **weight**, along which range the possible values for that property.

font family: A set of fonts that all have common stroke width and serif characteristics. For example, Times Roman and Times Roman Italic are members of the same font family.

gamma: A value that describes the way brightness is distributed across the intensity spectrum by a graphics device. Depending on the device, the gamma can 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 affects midrange tones. Gamma values are used to implement **gamma correction**. Typically, separate gamma values are used for each component of a color space.

gamma correction: In digital imaging, the process of changing the brightness, contrast, or color balance of an image by assigning new values (different colors) to gray or color tones.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

gradient line: The line in a color space along which color variation is greatest.

Graphics Device Interface (GDI): A Windows API, supported on 16-bit and 32-bit versions of the operating system, that supports graphics operations and image manipulation on logical graphics objects.

Graphics Device Interface, Extended (GDI+): A Windows API, supported on 32-bit and 64-bit versions of the operating system, that extends **GDI** to include support for Bezier curves, gradient brushes, image effects, and **EMF+** metafiles.

Graphics Interchange Format (GIF): A compression format that supports device-independent transmission and interchange of bitmapped image data. The format uses a palette of up to 256 distinct colors from the 24-bit **RGB** color space. It also supports animation and a separate palette of 256 colors for each frame. The color limitation makes the GIF format unsuitable for reproducing color photographs and other images with gradients of color, but it is well-suited for simpler images such as graphics with solid areas of color.

grayscale: A continuum of shades of gray that are 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 represent only a limited number of gray shades, 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 or images, to a representation with a discrete number of gray (or tone) levels.

highlight: The lightest tones in an image.

hotkey prefix: In a graphical user interface, the underlined letter in a word that can be pressed in combination with another key, such as the Alt key, to activate the functionality that the word represents.

hue: A color as defined by its name, such as blue. More precisely, hue is defined as the coordinates of a color in a color space, which specify the relative magnitudes along its axes.

hue saturation lightness effect: An **image effect** that is used to identify the basic color properties of an image, including **hue**, **saturation**, and **lightness**.

Image Color Management (ICM): Technology that ensures that 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.

image effect: A graphics process for changing the appearance of an image to produce a specific effect, including applying a **transform**, improving the quality of rendering, emphasizing or hiding a feature, creating a style, accounting for device limitations, and changing colors. The image effects specified in **EMF+ metafiles** include **blur**, brightness contrast, color balance, **color curve**, color lookup table, **color matrix**, hue saturation lightness, levels, **red-eye correction**, sharpen, and **tint**.

intensity: The magnitude of a component color in the color space (1).

Joint Photographic Experts Group (JPEG): A raster graphics file format for displaying high-resolution color graphics. JPEG graphics apply a user-specified compression scheme that can significantly reduce the file sizes of photo-realistic color graphics. A higher level of compression results in lower quality, whereas a lower level of compression results in higher quality. JPEG-format files have a .jpg or .jpeg file name extension.

levels effect: An **image effect** that is used to apply **highlight**, **midtone**, or **shadow** adjustments to an image. **Highlights** are the light parts of an image, **shadows** are the dark

parts, and **midtones** are the colors that occupy the middle of the tonal range between the **highlights** and the **shadows**.

lightness: The **brightness** or **intensity** of a color, from dark to light; or more precisely, the magnitude of the coordinates of a point in a color space.

line cap: The shape that is used at the end of a line drawn by a graphics pen.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

metafile: A sequence of record structures that store an image in an application-independent format. Metafile records contain drawing commands, object definitions, and configuration settings. When a metafile is processed, the stored image can be rendered on a display, output to a printer or plotter, stored in memory, or saved to a file or stream.

midtone: The tones in an image between **highlight** and **shadow**.

miter length: At the intersection of two lines, the distance from the intersection of the line walls on the inside of the line join to the intersection of the line walls on the outside of the line join. The miter length can be large when the angle between two lines is small. If the miter length of the join of an intersection exceeds a specified limit, the join can be beveled to keep it within the limit of the join of the intersection.

page space: A logical coordinate system used for graphics operations. It is determined by the mapping mode. Page space is defined with device-independent units, such as pixels.

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.

playback device context: The **device context** that defines the current graphics state during playback of the **metafile**. Although the data in a **metafile** can be device-independent, playback is always associated with an output device with specific properties, such as resolution, color support, and so on.

Portable Network Graphics (PNG): A bitmap graphics file format that uses lossless data compression and supports variable transparency of images (alpha channels) and control of image brightness on different computers (gamma correction). PNG-format files have a .png file name extension.

raster operation: The process of combining the bits in a source **bitmap** with the bits in a destination **bitmap** and in a specified pattern, to achieve a particular graphical output.

red-eye correction effect: An **image effect** that is used to correct the red eyes that sometimes occur in flash photographs as a result of the reflection of light from the flash.

red-green-blue (RGB): A color model that describes color information in terms of the red (R), green (G), and blue (B) intensities in a color.

reflection transform: A **transform** that is used to create a mirror image of an object with respect to either the horizontal or vertical axis.

region: A graphics object that is nonrectilinear in shape and is defined by an array of scanlines.

rotation: A transform that is used to rotate an object. When rotation occurs, the points that make up the object are rotated with respect to the coordinate space origin.

run-length encoding (RLE) compression: A form of data compression in which repeated values are represented by a count and a single instance of the value. RLE compression can significantly reduce disk and memory space requirements.

saturation: The "purity" of a **hue**; or, more precisely, the **intensity** of one **color channel** relative to the **intensity** of the other **color channels**. Maximum saturation occurs when the **intensity** of a particular **color channel** is maximum and the intensities of the other **color channels** are minimum. Minimum saturation occurs when the intensities of all **color channels** are the same.

scaling transform: A **transform** that is used to stretch or compress an object horizontally or vertically.

shadow: The darkest tones in an image.

sharpen effect: An **image effect** that is used to adjust the sharpness of an image. Sharpening increases image **contrast** by enhancing the definition of the image edges.

shear transform: A **transform** that is used to shear or cut an object. There are two components of a shear **transform**: The first alters the vertical lines in an object, and the second alters the horizontal lines.

spline: A sequence of individual curves joined to form a larger curve. A spline is specified by an array of points and a tension parameter.

Tag Image File Format (TIFF): A format for bitmapped image data that comes from scanners, frame grabbers, and photo-retouching applications. It supports the exchange of image data between applications, taking advantage of the varying capabilities of imaging devices. TIFF supports a number of compression schemes that allow the choice of the best space or time tradeoff for applications. For more information see [RFC3302] and [TIFF].

tent filter: A filtering algorithm in which pixels around a target pixel are weighted linearly based on their distance from the center of the target pixel.

terminal server: The computer on which nearly all 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.

text hinting: A mathematical process for adjusting the display of a font so that it lines up with a grid of pixels. At small screen sizes, hinting produces clearer text.

tint: The amount of a neutral color, such as black or white, that is mixed with another color. Changing the tint increases or decreases the **lightness** and **saturation**, and leaves the **hue** unchanged.

tint effect: An **image effect** that is used to apply a **tint** to an image. A **tint** is created by adding white to a color.

transform: An algorithm that transforms the size, orientation, and shape of objects that are copied from one **coordinate space** into another. Although a transform affects an object as a whole, it is applied to each point, or to each line, in the object.

translation transform: A **transform** that is used to shift each point in an object vertically, horizontally, or both, by a specified amount.

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 its design. Both TrueType and Type 1 font technologies are part of the OpenType format.

typeface: The primary design of a set of printed characters such as Courier, Helvetica, and Times Roman. The terms typeface and font are sometimes used interchangeably. A font is the particular implementation and variation of the typeface such as normal, bold, or italics. The distinguishing characteristic of a typeface is often the presence or absence of serifs.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

weight: The property of a font that specifies the degree of emphasis or boldness of the characters.

white saturation: The high end of the range of **saturation** for a **color channel**.

Windows metafile format (WMF): A file format used by Windows that supports the definition of images, including a format for clip art in word-processing documents.

world space: The most abstract logical **coordinate space** for graphics **transforms**. It allows **scaling**, **translation**, **rotation**, **shearing**, and **reflection**.

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.

[EXIF] Standard of Japan Electronics and Information Technology Industries Association (JEITA), "Exchangeable image File Format for Digital Still Cameras (EXIF)", Version 2.2, April, 2002, http://www.exif.org/Exif2-2.PDF

[GIF] Compuserve, Inc., "Graphics Interchange Format", 1989, http://www.piclist.com/techref/fileext/gif/gif89a.htm

[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://webstore.iec.ch/webstore/webstore.nsf/artnum/025408

[ISO/IEC-8859-1] International Organization for Standardization, "Information Technology -- 8-Bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1", ISO/IEC 8859-1, 1998, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=28245

Note There is a charge to download the specification.

[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".

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

[MS-WMF] Microsoft Corporation, "Windows Metafile Format".

[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.rfc-editor.org/rfc/rfc2119.txt

[RFC3302] Parsons, G., and Rafferty, J., "Tag Image File Format (TIFF) - image/tiff MIME Sub-Type Registration", RFC 3302, September 2002, http://www.ietf.org/rfc/rfc3302.txt

[TIFF] Adobe Developers Association, "TIFF 6.0 Specification", June 1992, http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf

[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

[MS-EMFSPOOL] Microsoft Corporation, "Enhanced Metafile Spool Format".

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference".

[MSDN-DrawBeziers] Microsoft Corporation, "Graphics.DrawBeziers(const Pen*, const PointF*, INT) method", http://msdn.microsoft.com/en-us/library/windows/desktop/ms536147(v=vs.85).aspx

[MSDN-GDI+] Microsoft Corporation, "GDI+", http://msdn.microsoft.com/en-us/library/ms533798.aspx

[OPENGL] Segal, M. and Akeley, K., "The OpenGL Graphics System: A Specification, Version 2.1", December 2006, http://www.opengl.org/registry/doc/glspec21.20061201.pdf

[PETZOLD] Petzold, C., "Programming Windows, Fifth Edition", Microsoft Press, 1998, ISBN: 157231995X.

[SPLINE77] Smith, A., "Spline Tutorial Notes - Technical Memo No. 77", SIGGRAPH '83 Tutorial Notes: Introduction to Computer Animation, pp. 64-75, July, 1983.

1.3 Overview

1.3.1 Metafile Structure

EMF+ defines a set of graphical images and text using commands, objects, and properties similar to Windows **GDI+** [MSDN-GDI+]. EMF+ metafiles are portable, device-independent containers for graphical images, and they are used for sending commands and objects to output devices, such as displays and printers, which support the drawing of images and text. The device or media that receives such a metafile can be located on a network, or it can be connected directly to the computer running the operating system on which the metafile is created.

EMF+ metafiles are actually a form of **EMF** metafile in which EMF+ records are embedded in EMF records ([MS-EMF] section 2.3). Embedding EMF+ records in EMF metafiles is possible because of the EMF capability to embed arbitrary private data in certain types of records. This is illustrated by the figure that follows. Note that multiple EMF+ records can be embedded in a single EMF record.

The EMF record in which arbitrary private data can be embedded is called an EMF "Comment" record. The form of EMF comment record that contains embedded EMF+ records is called EMR_COMMENT_EMFPLUS.

As shown in the following figure, the first EMF+ record in the metafile, the EMF+ **Header** record (section 2.3.3.3), is embedded within the first EMF record following the EMF **Header** record ([MS-EMF] section 2.3.4.2); and the last EMF+ record, the EMF+ **End of File** record (section 2.3.3.1), is embedded within the EMF record immediately preceding the EMF **End of File** record.

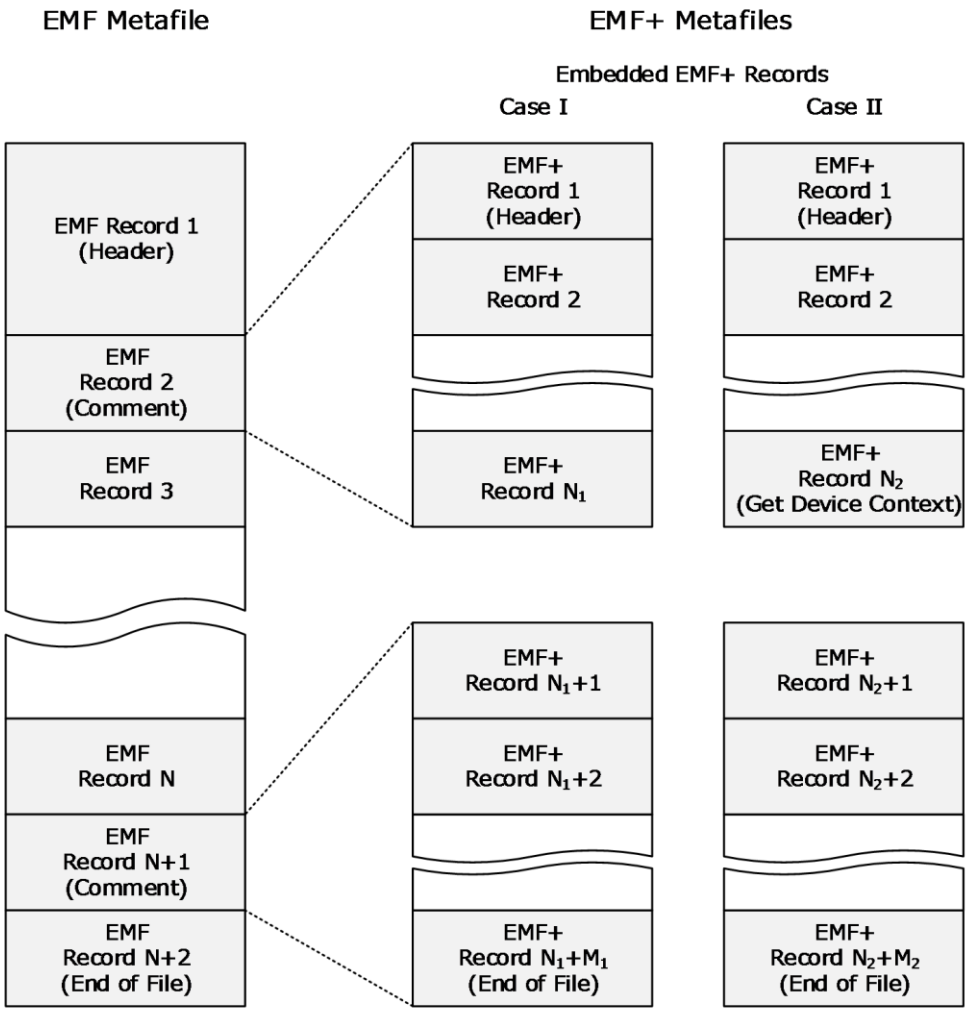


Figure 1: EMF+ metafiles

In the EMF metafile in this figure, Case I shows a group of EMF+ records, followed by some EMF records, followed by another group of EMF+ records; and Case II shows a group of EMF+ records terminated by a **Get Device Context** record (section 2.3.3.2), followed by some EMF records, followed by another group of EMF+ records. The presence or absence of the Get Device Context record can determine how the metafile is processed.

The structure of an EMF+ metafile is such that EMF+ records are embedded in EMF records, meaning that some outer EMF records are always present, namely EMF control records, and EMF records that contain EMF+ records:

- EMR_HEADER - required for all types of metafiles
- EMR_COMMENT – required to contain EMF+ records
- EMR_EOF - required to terminate all types of metafiles

The EMF+ **Header** record contains flags that distinguish between two different types of EMF+ metafile.<1>

- Metafiles identified as **EMF+ Only** can contain both EMF+ records and EMF records. All EMF+ records are used to render the image. The EMF records that are part of the drawing are those preceded by a **Get Device Context** record—case II in the figure above. EMF record processing stops when the next EMF+ record of any type is encountered.

  If a system that cannot play back EMF+ records attempts to play the metafile by using only EMF records, the drawing might be incomplete. If a system performs EMF+ playback mode as expected, then no EMF drawing records are processed unless they are preceded by a Get Device Context record. For example, in case I, an **EMF+ Only** playback mode would process the EMF control records and none of the EMF drawing records. As a result, EMF records alone do not suffice to render the drawing that was recorded in an **EMF+ Only** metafile.

- Metafiles identified as **EMF+ Dual** can also contain both EMF+ records and EMF records. The **EMF+ Dual** flag indicates that the metafile contains a complete set of EMF records sufficient to correctly render the entire drawing on a system that cannot playback EMF+ records. This feature makes it possible to render an image with different levels of graphics support in the operating system. However, only one or the other type of records is processed. All records are enumerated sequentially. For EMF playback, the metafile player only uses EMF records and ignores EMF+ records. For EMF+ playback, the metafile is played as if it is **EMF+ Only**.

For either type of EMF+ metafile, the EMF records that follow an EmfPlusGetDC record are played, until the next EMF+ record, EMF_HEADER, or EMF_EOF (), regardless of the **EMF+ Dual** flag setting.

Note: EMF+ is not considered an extension to the EMF feature set; that is, EMF+ does not define new EMF records. EMF+ is semantically a completely separate, independent format. EMF+ records define graphical images and text using commands, objects, and properties of GDI+.

1.3.2 Byte Ordering

Data in the EMF+ metafile records are 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 byte numbering used for bitfields in this specification is big-endian. Other architectures number the bytes in a binary word from right to left, which is referred to as little-endian. The byte numbering used for enumerations, objects, and records in this specification is little-endian.

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

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 Structures

The following formats define metafile structures that are directly or indirectly related to the EMF+ metafile structure:

- Enhanced metafile format (EMF) [MS-EMF] is the predecessor to EMF+. EMF metafiles can contain EMF+ metafiles.
- **Enhanced metafile spool format (EMFSPOOL)** [MS-EMFSPOOL] metafiles can contain EMF metafiles.
- **Windows metafile format (WMF)** [MS-WMF] is the 16-bit predecessor to EMF. WMF metafiles can be embedded in EMF+ metafiles.

This is illustrated by the following figure:

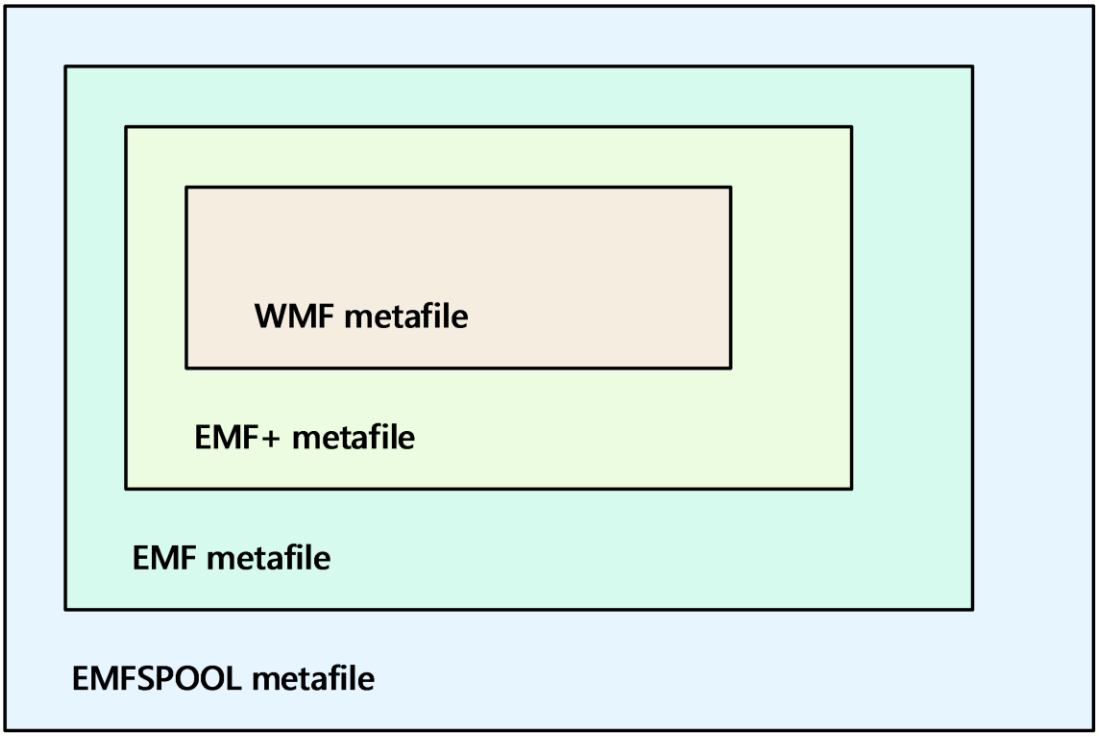


Figure 2: Metafile nesting

1.5 Applicability Statement

Structures that are compliant with the EMF+ can be used as portable, system-independent containers for images. The graphics supported in EMF+ metafiles are applicable to document content representation, including printing and plotting.

1.6 Versioning and Localization

This specification covers versioning issues in the following areas:

Structure Versions: Multiple versions of the EMF+ metafiles exist. For more information, see the GraphicsVersion enumeration.

Localization: EMF+ structures define the following locale-specific data:

- Language identifiers that correspond to natural languages in locales, including countries, geographical regions, and administrative districts. For more information, see the LanguageIdentifier enumeration. For information concerning Windows version support for language identifiers, see [MS-LCID].
- String digit substitution values specify how to substitute digits in a string according to a locale or language. For more information, see the StringDigitSubstitute enumeration.

1.7 Vendor-Extensible Fields

The EMF+ metafile format includes the following vendor-extensible fields:

- Arbitrary private data in an EmfPlusComment record
- The graphics version in an EmfPlusGraphicsVersion object
- The language identifier in an EmfPlusLanguageIdentifier object
- Digit substitution in an EmfPlusStringFormat object

2 Structures

This section specifies the EMF+ structures and how they are serialized in a metafile. EMF+ structures are grouped into the following categories:

Name	Section	Description
EMF+ Constants	2.1	Specify enumerations, bit flags, and standard identifiers.
EMF+ Objects	2.2	Specify graphics objects, structure objects, and image effects objects.
EMF+ Records	2.3	Specify the format of metafile records, which define graphics operations and manage the **playback device context**.

This protocol references commonly used data types as defined in [MS-DTYP].

2.1 EMF+ Constants

This section specifies the EMF+ Constants, which are grouped into the following categories:

Name	Section	Description
Enumeration constant types	2.1.1	Specify numeric constants used in EMF+ objects and records.
Bit Flag constant types	2.1.2	Specify properties and options for EMF+ objects and records. In general, bit flags can be combined with Boolean OR operations.
Standard Identifier constant types	2.1.3	Specify **globally unique identifiers (GUIDs)** for EMF+ objects and records.

2.1.1 Enumeration Constant Types

The EMF+ Enumeration Constants specify numeric constants that are used in EMF+ objects and records.

The following types of enumerations are defined:

Name	Section	Description
RecordType	2.1.1.1	Defines record types used in EMF+ metafiles.
BitmapDataType	2.1.1.2	Defines types of **bitmap** data formats.
BrushType	2.1.1.3	Defines types of graphics brushes, which are used to fill graphics regions.
CombineMode	2.1.1.4	Defines modes for combining two graphics regions.
CompositingMode	2.1.1.5	Defines modes for combining source colors with background colors. The **compositing** mode represents the enable state of **alpha blending**.
CompositingQuality	2.1.1.6	Defines levels of quality for creating composite images.
CurveAdjustments	2.1.1.7	Defines **color curve effects** that can be applied to an image.
CurveChannel	2.1.1.8	Defines **color channels** that can be affected by a **color curve effect**

Name	Section	Description
		adjustment to an image.
CustomLineCapDataType	2.1.1.9	Defines types of custom **line cap** data, which specify styles and shapes for the ends of graphics lines.
DashedLineCapType	2.1.1.10	Defines types of line caps to use at the ends of dashed lines that are drawn with graphics pens.
FilterType	2.1.1.11	Defines types of filtering algorithms that can be used for text and graphics quality enhancement and image rendering.
GraphicsVersion	2.1.1.12	Defines versions of operating system graphics that are used to create EMF+ metafiles.
HatchStyle	2.1.1.13	Defines hatch patterns used by graphics brushes. A hatch pattern consists of a solid background color and lines drawn over the background.
HotkeyPrefix	2.1.1.14	Defines output options for **hotkey prefixes** in graphics text.
ImageDataType	2.1.1.15	Defines types of image data formats.
InterpolationMode	2.1.1.16	Defines ways to perform **scaling**, including stretching and shrinking.
LanguageIdentifier	2.1.1.17	Defines identifiers for natural languages in locales, including countries, geographical regions, and administrative districts.
LineCapType	2.1.1.18	Defines types of line caps to use at the ends of lines that are drawn with graphics pens.
LineJoinType	2.1.1.19	Defines ways to join two lines that are drawn by the same graphics pen and whose ends meet.
LineStyle	2.1.1.20	Defines styles of lines that are drawn with graphics pens.
MetafileDataType	2.1.1.21	Defines types of metafiles data that can be embedded in an EMF+ metafile.
ObjectType	2.1.1.22	Defines types of graphics objects that can be created and used in graphics operations.
PathPointType	2.1.1.23	Defines types of points on a graphics **path**.
PenAlignment	2.1.1.24	Defines the distribution of the width of the pen with respect to the line being drawn.
PixelFormat	2.1.1.25	Defines pixel formats that are supported in EMF+ bitmaps.
PixelOffsetMode	2.1.1.26	Defines how pixels are offset, which specifies the trade-off between rendering speed and quality.
RegionNodeDataType	2.1.1.27	Defines types of **region** node data.
SmoothingMode	2.1.1.28	Defines types of smoothing to apply to lines, curves and the edges of filled areas to make them appear more continuous or sharply defined.
StringAlignment	2.1.1.29	Defines ways to align strings with respect to a text layout rectangle.
StringDigitSubstitution	2.1.1.30	Defines ways to substitute digits in a string according to a user's locale or language.
StringTrimming	2.1.1.31	Defines how to trim characters from a string that is too large for the text layout rectangle.

Name	Section	Description
TextRenderingHint	2.1.1.32	Defines types of **text hinting** and **anti-aliasing**, which affects the quality of text rendering.
UnitType	2.1.1.33	Defines units of measurement in different coordinate systems.

2.1.1.1 RecordType Enumeration

The RecordType enumeration defines record types used in EMF+ metafiles.

```
typedef  enum
{
  EmfPlusHeader = 0x4001,
  EmfPlusEndOfFile = 0x4002,
  EmfPlusComment = 0x4003,
  EmfPlusGetDC = 0x4004,
  EmfPlusMultiFormatStart = 0x4005,
  EmfPlusMultiFormatSection = 0x4006,
  EmfPlusMultiFormatEnd = 0x4007,
  EmfPlusObject = 0x4008,
  EmfPlusClear = 0x4009,
  EmfPlusFillRects = 0x400A,
  EmfPlusDrawRects = 0x400B,
  EmfPlusFillPolygon = 0x400C,
  EmfPlusDrawLines = 0x400D,
  EmfPlusFillEllipse = 0x400E,
  EmfPlusDrawEllipse = 0x400F,
  EmfPlusFillPie = 0x4010,
  EmfPlusDrawPie = 0x4011,
  EmfPlusDrawArc = 0x4012,
  EmfPlusFillRegion = 0x4013,
  EmfPlusFillPath = 0x4014,
  EmfPlusDrawPath = 0x4015,
  EmfPlusFillClosedCurve = 0x4016,
  EmfPlusDrawClosedCurve = 0x4017,
  EmfPlusDrawCurve = 0x4018,
  EmfPlusDrawBeziers = 0x4019,
  EmfPlusDrawImage = 0x401A,
  EmfPlusDrawImagePoints = 0x401B,
  EmfPlusDrawString = 0x401C,
  EmfPlusSetRenderingOrigin = 0x401D,
  EmfPlusSetAntiAliasMode = 0x401E,
  EmfPlusSetTextRenderingHint = 0x401F,
  EmfPlusSetTextContrast = 0x4020,
  EmfPlusSetInterpolationMode = 0x4021,
  EmfPlusSetPixelOffsetMode = 0x4022,
  EmfPlusSetCompositingMode = 0x4023,
  EmfPlusSetCompositingQuality = 0x4024,
  EmfPlusSave = 0x4025,
  EmfPlusRestore = 0x4026,
  EmfPlusBeginContainer = 0x4027,
  EmfPlusBeginContainerNoParams = 0x4028,
  EmfPlusEndContainer = 0x4029,
  EmfPlusSetWorldTransform = 0x402A,
  EmfPlusResetWorldTransform = 0x402B,
  EmfPlusMultiplyWorldTransform = 0x402C,
  EmfPlusTranslateWorldTransform = 0x402D,
  EmfPlusScaleWorldTransform = 0x402E,
  EmfPlusRotateWorldTransform = 0x402F,
  EmfPlusSetPageTransform = 0x4030,
  EmfPlusResetClip = 0x4031,
  EmfPlusSetClipRect = 0x4032,
  EmfPlusSetClipPath = 0x4033,
```

```
    EmfPlusSetClipRegion = 0x4034,
    EmfPlusOffsetClip = 0x4035,
    EmfPlusDrawDriverstring = 0x4036,
    EmfPlusStrokeFillPath = 0x4037,
    EmfPlusSerializableObject = 0x4038,
    EmfPlusSetTSGraphics = 0x4039,
    EmfPlusSetTSClip = 0x403A
} RecordType;
```

EmfPlusHeader: This record specifies the start of EMF+ data in the metafile. It MUST be embedded in the first EMF record after the EMF Header record.

EmfPlusEndOfFile: This record specifies the end of EMF+ data in the metafile.

EmfPlusComment: This record specifies arbitrary private data.

EmfPlusGetDC: This record specifies that subsequent EMF records ([MS-EMF] section 2.3) encountered in the metafile SHOULD be processed. EMF records cease being processed when the next EMF+ record is encountered.

EmfPlusMultiFormatStart: This record is reserved and MUST NOT be used.

EmfPlusMultiFormatSection: This record is reserved and MUST NOT be used.

EmfPlusMultiFormatEnd: This record is reserved and MUST NOT be used.

EmfPlusObject: This record specifies an object for use in graphics operations.

EmfPlusClear: This record clears the output **coordinate space** and initializes it with a specified background color and transparency.

EmfPlusFillRects: This record defines how to fill the interiors of a series of rectangles, using a specified brush.

EmfPlusDrawRects: This record defines the pen strokes for drawing a series of rectangles.

EmfPlusFillPolygon: This record defines the data to fill the interior of a polygon, using a specified brush.

EmfPlusDrawLines: This record defines the pen strokes for drawing a series of connected lines.

EmfPlusFillEllipse: This record defines how to fill the interiors of an ellipse, using a specified brush.

EmfPlusDrawEllipse: This record defines the pen strokes for drawing an ellipse.

EmfPlusFillPie: This record defines how to fill a section of an interior section of an ellipse using a specified brush.

EmfPlusDrawPie: This record defines pen strokes for drawing a section of an ellipse.

EmfPlusDrawArc: The record defines pen strokes for drawing an arc of an ellipse.

EmfPlusFillRegion: This record defines how to fill the interior of a region using a specified brush.

EmfPlusFillPath: The record defines how to fill the interiors of the figures defined in a graphics path with a specified brush. A path is an object that defines an arbitrary sequence of lines, curves, and shapes.

EmfPlusDrawPath: The record defines the pen strokes to draw the figures in a graphics path. A path is an object that defines an arbitrary sequence of lines, curves, and shapes.

EmfPlusFillClosedCurve: This record defines how to fill the interior of a **closed cardinal spline** using a specified brush.

EmfPlusDrawClosedCurve: This record defines the pen and strokes for drawing a closed cardinal spline.

EmfPlusDrawCurve: This record defines the pen strokes for drawing a **cardinal spline**.

EmfPlusDrawBeziers: This record defines the pen strokes for drawing a **Bezier spline**.

EmfPlusDrawImage: This record defines a scaled EmfPlusImage object. An image can consist of either bitmap or metafile data.

EmfPlusDrawImagePoints: This record defines a scaled EmfPlusImage object inside a parallelogram. An image can consist of either bitmap or metafile data.

EmfPlusDrawString: This record defines a text string based on a font, a layout rectangle, and a format.

EmfPlusSetRenderingOrigin: This record defines the origin of rendering to the specified horizontal and vertical coordinates. This applies to hatch brushes and to 8 and 16 bits per pixel dither patterns.

EmfPlusSetAntiAliasMode: This record defines whether to enable or disable text anti-aliasing. Text anti-aliasing is a method of making lines and edges of character glyphs appear smoother when drawn on an output surface.

EmfPlusSetTextRenderingHint: This record defines the process used for rendering text.

EmfPlusSetTextContrast: This record sets text **contrast** according to the specified text **gamma** value.

EmfPlusSetInterpolationMode: This record defines the interpolation mode of an object according to the specified type of image filtering. The interpolation mode influences how scaling (stretching and shrinking) is performed.

EmfPlusSetPixelOffsetMode: This record defines the pixel offset mode according to the specified pixel centering value.

EmfPlusSetCompositingMode: This record defines the compositing mode according to the state of alpha blending, which specifies how source colors are combined with background colors.

EmfPlusSetCompositingQuality: This record defines the compositing quality, which describes the desired level of quality for creating composite images from multiple objects.

EmfPlusSave: This record saves the graphics state, identified by a specified index, on a stack of saved graphics states. Each stack index is associated with a particular saved state, and the index is used by an EmfPlusRestore record to restore the state.

EmfPlusRestore: This record restores the graphics state, identified by a specified index, from a stack of saved graphics states. Each stack index is associated with a particular saved state, and the index is defined by an EmfPlusSave record to save the state.

EmfPlusBeginContainer: This record opens a new graphics state container and specifies a **transform** for it. Graphics containers are used to retain elements of the graphics state.

EmfPlusBeginContainerNoParams: This record opens a new graphics state container.

EmfPlusEndContainer: This record closes a graphics state container that was previously opened by a begin container operation.

EmfPlusSetWorldTransform: This record defines the current **world space** transform in the playback device context, according to a specified transform matrix.

EmfPlusResetWorldTransform: This record resets the current world space transform to the identify matrix.

EmfPlusMultiplyWorldTransform: This record multiplies the current world space by a specified transform matrix.

EmfPlusTranslateWorldTransform: This record applies a **translation transform** to the current world space by specified horizontal and vertical distances.

EmfPlusScaleWorldTransform: This record applies a scaling transform to the current world space by specified horizontal and vertical scale factors.

EmfPlusRotateWorldTransform: This record rotates the current world space by a specified angle.

EmfPlusSetPageTransform: This record specifies extra scaling factors for the current world space transform.

EmfPlusResetClip: This record resets the current clipping region for the world space to infinity.

EmfPlusSetClipRect: This record combines the current clipping region with a rectangle.

EmfPlusSetClipPath: This record combines the current clipping region with a graphics path.

EmfPlusSetClipRegion: This record combines the current clipping region with another graphics region.

EmfPlusOffsetClip: This record applies a translation transform on the current clipping region of the world space.

EmfPlusDrawDriverstring: This record specifies text output with character positions.

EmfPlusStrokeFillPath: This record closes any open figures in a path, strokes the outline of the path by using the current pen, and fills its interior by using the current brush.

EmfPlusSerializableObject: This record defines an **image effects** parameter block that has been serialized into a data buffer.

EmfPlusSetTSGraphics: This record specifies the state of a graphics **device context** for a **terminal server**.

EmfPlusSetTSClip: This record specifies clipping areas in the graphics device context for a terminal server.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.2 BitmapDataType Enumeration

The BitmapDataType enumeration defines types of bitmap data formats.

```
typedef  enum
{
  BitmapDataTypePixel = 0x00000000,
  BitmapDataTypeCompressed = 0x00000001
} BitmapDataType;
```

BitmapDataTypePixel: Specifies a bitmap image with pixel data.

BitmapDataTypeCompressed: Specifies an image with compressed data.

Bitmap data is specified by EmfPlusBitmap objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.3 BrushType Enumeration

The BrushType enumeration defines types of graphics brushes, which are used to fill graphics regions.

```
typedef  enum
{
  BrushTypeSolidColor = 0x00000000,
  BrushTypeHatchFill = 0x00000001,
  BrushTypeTextureFill = 0x00000002,
  BrushTypePathGradient = 0x00000003,
  BrushTypeLinearGradient = 0x00000004
} BrushType;
```

BrushTypeSolidColor: Specifies a solid-color brush, which is characterized by an EmfPlusARGB value.

BrushTypeHatchFill: Specifies a hatch brush, which is characterized by a predefined pattern.

BrushTypeTextureFill: Specifies a texture brush, which is characterized by an image.

BrushTypePathGradient: Specifies a path gradient brush, which is characterized by a color gradient path gradient brush data.

BrushTypeLinearGradient: **BrushData** contains linear gradient brush data.

Graphics brushes are specified by EmfPlusBrush objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.4 CombineMode Enumeration

The CombineMode enumeration defines modes for combining two graphics regions. In the following descriptions, the regions to be combined are referred to as the "existing" and "new" regions.

```
typedef  enum
{
  CombineModeReplace = 0x00000000,
  CombineModeIntersect = 0x00000001,
  CombineModeUnion = 0x00000002,
  CombineModeXOR = 0x00000003,
  CombineModeExclude = 0x00000004,
  CombineModeComplement = 0x00000005
} CombineMode;
```

CombineModeReplace: Replaces the existing region with the new region.

CombineModeIntersect: Replaces the existing region with the intersection of the existing region and the new region.

CombineModeUnion: Replaces the existing region with the union of the existing and new regions.

CombineModeXOR: Replaces the existing region with the XOR of the existing and new regions.

CombineModeExclude: Replaces the existing region with the part of itself that is not in the new region.

CombineModeComplement: Replaces the existing region with the part of the new region that is not in the existing region.

Graphics regions are specified by EmfPlusRegion objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.5 CompositingMode Enumeration

The CompositingMode enumeration defines modes for combining source colors with background colors. The compositing mode represents the enable state of alpha blending.

```
typedef  enum
{
  CompositingModeSourceOver = 0x00,
  CompositingModeSourceCopy = 0x01
} CompositingMode;
```

CompositingModeSourceOver: Enables alpha blending, which specifies that when a color is rendered, it is blended with the background color. The extent of blending is determined by the value of the **alpha** component of the color being rendered.

CompositingModeSourceCopy: Disables alpha blending, which means that when a source color is rendered, it overwrites the background color.

Graphics colors are specified by EmfPlusARGB objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.6 CompositingQuality Enumeration

The CompositingQuality enumeration defines levels of quality for creating composite images.<2>

```
typedef  enum
{
  CompositingQualityDefault = 0x01,
  CompositingQualityHighSpeed = 0x02,
  CompositingQualityHighQuality = 0x03,
  CompositingQualityGammaCorrected = 0x04,
  CompositingQualityAssumeLinear = 0x05
} CompositingQuality;
```

CompositingQualityDefault: No **gamma correction** is performed. Gamma correction controls the overall **brightness** and contrast of an image. Without gamma correction, composited images can appear too light or too dark.

CompositingQualityHighSpeed: No gamma correction is performed. Compositing speed is favored at the expense of quality. In terms of the result, there is no difference between this value and CompositingQualityDefault.

CompositingQualityHighQuality: Gamma correction is performed. Compositing quality is favored at the expense of speed.

CompositingQualityGammaCorrected: Enable gamma correction for higher-quality compositing with lower speed. In terms of the result, there is no difference between this value and CompositingQualityHighQuality.

CompositingQualityAssumeLinear: No gamma correction is performed; however, using linear values results in better quality than the default at a slightly lower speed.

Graphics colors are specified by EmfPlusARGB objects.

Compositing is done during rendering when source pixels are combined with destination pixels. The compositing quality directly relates to the visual quality of the output and is inversely proportional to the time required for rendering. The higher the quality, the more surrounding pixels need to be taken into account during the compositing operation; hence, the slower the render time.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.7 CurveAdjustments Enumeration

The CurveAdjustments enumeration defines adjustments that can be applied to the color curve of an image.

```
typedef  enum
{
  AdjustExposure = 0x00000000,
  AdjustDensity = 0x00000001,
  AdjustContrast = 0x00000002,
  AdjustHighlight = 0x00000003,
  AdjustShadow = 0x00000004,
  AdjustMidtone = 0x00000005,
  AdjustWhiteSaturation = 0x00000006,
  AdjustBlackSaturation = 0x00000007
} CurveAdjustments;
```

AdjustExposure: Specifies the simulation of increasing or decreasing the **exposure** of an image.

AdjustDensity: Specifies the simulation of increasing or decreasing the **density** of an image.

AdjustContrast: Specifies an increase or decrease of the contrast of an image.

AdjustHighlight: Specifies an increase or decrease of the value of a color channel of an image, if that channel already has a value that is above half **intensity**. This adjustment can be used to increase definition in the light areas of an image without affecting the dark areas.

AdjustShadow: Specifies an increase or decrease of the value of a color channel of an image, if that channel already has a value that is below half intensity. This adjustment can be used to increase definition in the dark areas of an image without affecting the light areas.

AdjustMidtone: Specifies an adjustment that lightens or darkens an image. Color channel values in the middle of the intensity range are altered more than color channel values near the minimum or maximum extremes of intensity. This adjustment can be used to lighten or darken an image without losing the contrast between the darkest and lightest parts of the image.

AdjustWhiteSaturation: Specifies an adjustment to the **white saturation** of an image, defined as the maximum value in the range of intensities for a given color channel, whose range is typically 0 to 255.

For example, a white saturation adjustment value of 240 specifies that color channel values in the range 0 to 240 are adjusted so that they spread out over the range 0 to 255, with color channel values greater than 240 set to 255.

AdjustBlackSaturation: Specifies an adjustment to the **black saturation** of an image, which is the minimum value in the range of intensities for a given color channel, which is typically 0 to 255.

For example, a black saturation adjustment value of 15 specifies that color channel values in the range 15 to 255 are adjusted so that they spread out over the range 0 to 255, with color channel values less than 15 set to 0.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.8 CurveChannel Enumeration

The CurveChannel enumeration defines color channels that can be affected by a color curve effect adjustment to an image.

```
typedef  enum
{
  CurveChannelAll = 0x00000000,
  CurveChannelRed = 0x00000001,
  CurveChannelGreen = 0x00000002,
  CurveChannelBlue = 0x00000003
} CurveChannel;
```

CurveChannelAll: Specifies that a color curve adjustment applies to all color channels.

CurveChannelRed: Specifies that a color curve adjustment applies only to the red color channel.

CurveChannelGreen: Specifies that a color curve adjustment applies only to the green color channel.

CurveChannelBlue: Specifies that a color curve adjustment applies only to the blue color channel.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.9 CustomLineCapDataType Enumeration

The CustomLineCapDataType enumeration defines types of custom line cap data, which specify styles and shapes for the ends of graphics lines.

```
typedef  enum
{
  CustomLineCapDataTypeDefault = 0x00000000,
  CustomLineCapDataTypeAdjustableArrow = 0x00000001
} CustomLineCapDataType;
```

CustomLineCapDataTypeDefault: Specifies a default custom line cap.

CustomLineCapDataTypeAdjustableArrow: Specifies an adjustable arrow custom line cap.

Custom line cap data is specified by EmfPlusCustomLineCap objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.10 DashedLineCapType Enumeration

The DashedLineCapType enumeration defines types of line caps to use at the ends of dashed lines that are drawn with graphics pens.

```
typedef  enum
{
  DashedLineCapTypeFlat = 0x00000000,
  DashedLineCapTypeRound = 0x00000002,
  DashedLineCapTypeTriangle = 0x00000003
} DashedLineCapType;
```

DashedLineCapTypeFlat: Specifies a flat dashed line cap.

DashedLineCapTypeRound: Specifies a round dashed line cap.

DashedLineCapTypeTriangle: Specifies a triangular dashed line cap.

Dashed lines are specified by EmfPlusDashedLineData objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.11 FilterType Enumeration

The FilterType enumeration defines types of filtering algorithms that can be used for text and graphics quality enhancement and image rendering.

```
typedef  enum
{
  FilterTypeNone = 0x00,
  FilterTypePoint = 0x01,
  FilterTypeLinear = 0x02,
  FilterTypeTriangle = 0x03,
  FilterTypeBox = 0x04,
  FilterTypePyramidalQuad = 0x06,
  FilterTypeGaussianQuad = 0x07
} FilterType;
```

FilterTypeNone: Specifies that filtering is not performed.

FilterTypePoint: Specifies that each destination pixel is computed by sampling the nearest pixel from the source image.

FilterTypeLinear: Specifies that linear interpolation is performed using the weighted average of a 2x2 area of pixels surrounding the source pixel.

FilterTypeTriangle: Specifies that each pixel in the source image contributes equally to the destination image. This is the slowest of filtering algorithms.

FilterTypeBox: Specifies a **box filter** algorithm, in which each destination pixel is computed by averaging a rectangle of source pixels. This algorithm is useful only when reducing the size of an image.

FilterTypePyramidalQuad: Specifies that a 4-sample **tent filter** is used.

FilterTypeGaussianQuad: Specifies that a 4-sample Gaussian filter is used, which creates a **blur effect** on an image.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.12 GraphicsVersion Enumeration

The GraphicsVersion enumeration defines versions of operating system graphics that are used to create EMF+ metafiles.

```
typedef  enum
{
  GraphicsVersion1 = 0x0001,
  GraphicsVersion1_1 = 0x0002
} GraphicsVersion;
```

GraphicsVersion1: Specifies GDI+ version 1.0.

GraphicsVersion1_1: Specifies GDI+ version 1.1.<3>

Graphics versions are specified in EmfPlusGraphicsVersion objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.13 HatchStyle Enumeration

The HatchStyle enumeration defines hatch patterns used by graphics brushes. A hatch pattern consists of a solid background color and lines drawn over the background.

```
typedef  enum
{
  HatchStyleHorizontal = 0x00000000,
  HatchStyleVertical = 0x00000001,
  HatchStyleForwardDiagonal = 0x00000002,
  HatchStyleBackwardDiagonal = 0x00000003,
  HatchStyleLargeGrid = 0x00000004,
  HatchStyleDiagonalCross = 0x00000005,
  HatchStyle05Percent = 0x00000006,
  HatchStyle10Percent = 0x00000007,
  HatchStyle20Percent = 0x00000008,
  HatchStyle25Percent = 0x00000009,
  HatchStyle30Percent = 0x0000000A,
  HatchStyle40Percent = 0x0000000B,
  HatchStyle50Percent = 0x0000000C,
  HatchStyle60Percent = 0x0000000D,
  HatchStyle70Percent = 0x0000000E,
  HatchStyle75Percent = 0x0000000F,
  HatchStyle80Percent = 0x00000010,
  HatchStyle90Percent = 0x00000011,
  HatchStyleLightDownwardDiagonal = 0x00000012,
  HatchStyleLightUpwardDiagonal = 0x00000013,
  HatchStyleDarkDownwardDiagonal = 0x00000014,
  HatchStyleDarkUpwardDiagonal = 0x00000015,
  HatchStyleWideDownwardDiagonal = 0x00000016,
  HatchStyleWideUpwardDiagonal = 0x00000017,
  HatchStyleLightVertical = 0x00000018,
  HatchStyleLightHorizontal = 0x00000019,
  HatchStyleNarrowVertical = 0x0000001A,
  HatchStyleNarrowHorizontal = 0x0000001B,
  HatchStyleDarkVertical = 0x0000001C,
  HatchStyleDarkHorizontal = 0x0000001D,
  HatchStyleDashedDownwardDiagonal = 0x0000001E,
  HatchStyleDashedUpwardDiagonal = 0x0000001F,
  HatchStyleDashedHorizontal = 0x00000020,
  HatchStyleDashedVertical = 0x00000021,
  HatchStyleSmallConfetti = 0x00000022,
  HatchStyleLargeConfetti = 0x00000023,
  HatchStyleZigZag = 0x00000024,
  HatchStyleWave = 0x00000025,
```

```
  HatchStyleDiagonalBrick = 0x00000026,
  HatchStyleHorizontalBrick = 0x00000027,
  HatchStyleWeave = 0x00000028,
  HatchStylePlaid = 0x00000029,
  HatchStyleDivot = 0x0000002A,
  HatchStyleDottedGrid = 0x0000002B,
  HatchStyleDottedDiamond = 0x0000002C,
  HatchStyleShingle = 0x0000002D,
  HatchStyleTrellis = 0x0000002E,
  HatchStyleSphere = 0x0000002F,
  HatchStyleSmallGrid = 0x00000030,
  HatchStyleSmallCheckerBoard = 0x00000031,
  HatchStyleLargeCheckerBoard = 0x00000032,
  HatchStyleOutlinedDiamond = 0x00000033,
  HatchStyleSolidDiamond = 0x00000034
} HatchStyle;
```

HatchStyleHorizontal: Specifies equally spaced horizontal lines.

HatchStyleVertical: Specifies equally spaced vertical lines.

HatchStyleForwardDiagonal: Specifies lines on a diagonal from upper left to lower right.

HatchStyleBackwardDiagonal: Specifies lines on a diagonal from upper right to lower left.

HatchStyleLargeGrid: Specifies crossing horizontal and vertical lines.

HatchStyleDiagonalCross: Specifies crossing forward diagonal and backward diagonal lines with anti-aliasing.

HatchStyle05Percent: Specifies a 5-percent hatch, which is the ratio of foreground color to background color equal to 5:100.

HatchStyle10Percent: Specifies a 10-percent hatch, which is the ratio of foreground color to background color equal to 10:100.

HatchStyle20Percent: Specifies a 20-percent hatch, which is the ratio of foreground color to background color equal to 20:100.

HatchStyle25Percent: Specifies a 25-percent hatch, which is the ratio of foreground color to background color equal to 25:100.

HatchStyle30Percent: Specifies a 30-percent hatch, which is the ratio of foreground color to background color equal to 30:100.

HatchStyle40Percent: Specifies a 40-percent hatch, which is the ratio of foreground color to background color equal to 40:100.

HatchStyle50Percent: Specifies a 50-percent hatch, which is the ratio of foreground color to background color equal to 50:100.

HatchStyle60Percent: Specifies a 60-percent hatch, which is the ratio of foreground color to background color equal to 60:100.

HatchStyle70Percent: Specifies a 70-percent hatch, which is the ratio of foreground color to background color equal to 70:100.

HatchStyle75Percent: Specifies a 75-percent hatch, which is the ratio of foreground color to background color equal to 75:100.

HatchStyle80Percent: Specifies an 80-percent hatch, which is the ratio of foreground color to background color equal to 80:100.

HatchStyle90Percent: Specifies a 90-percent hatch, which is the ratio of foreground color to background color equal to 90:100.

HatchStyleLightDownwardDiagonal: Specifies diagonal lines that slant to the right from top to bottom points with no anti-aliasing. They are spaced 50 percent further apart than lines in the HatchStyleForwardDiagonal pattern

HatchStyleLightUpwardDiagonal: Specifies diagonal lines that slant to the left from top to bottom points with no anti-aliasing. They are spaced 50 percent further apart than lines in the HatchStyleBackwardDiagonal pattern.

HatchStyleDarkDownwardDiagonal: Specifies diagonal lines that slant to the right from top to bottom points with no anti-aliasing. They are spaced 50 percent closer and are twice the width of lines in the HatchStyleForwardDiagonal pattern.

HatchStyleDarkUpwardDiagonal: Specifies diagonal lines that slant to the left from top to bottom points with no anti-aliasing. They are spaced 50 percent closer and are twice the width of lines in the HatchStyleBackwardDiagonal pattern.

HatchStyleWideDownwardDiagonal: Specifies diagonal lines that slant to the right from top to bottom points with no anti-aliasing. They have the same spacing between lines in HatchStyleWideDownwardDiagonal pattern and HatchStyleForwardDiagonal pattern, but HatchStyleWideDownwardDiagonal has the triple line width of HatchStyleForwardDiagonal.

HatchStyleWideUpwardDiagonal: Specifies diagonal lines that slant to the left from top to bottom points with no anti-aliasing. They have the same spacing between lines in HatchStyleWideUpwardDiagonal pattern and HatchStyleBackwardDiagonal pattern, but HatchStyleWideUpwardDiagonal has the triple line width of HatchStyleWideUpwardDiagonal.

HatchStyleLightVertical: Specifies vertical lines that are spaced 50 percent closer together than lines in the HatchStyleVertical pattern.

HatchStyleLightHorizontal: Specifies horizontal lines that are spaced 50 percent closer than lines in the HatchStyleHorizontal pattern.

HatchStyleNarrowVertical: Specifies vertical lines that are spaced 75 percent closer than lines in the HatchStyleVertical pattern; or 25 percent closer than lines in the HatchStyleLightVertical pattern.

HatchStyleNarrowHorizontal: Specifies horizontal lines that are spaced 75 percent closer than lines in the HatchStyleHorizontal pattern; or 25 percent closer than lines in the HatchStyleLightHorizontal pattern.

HatchStyleDarkVertical: Specifies lines that are spaced 50 percent closer than lines in the HatchStyleVertical pattern.

HatchStyleDarkHorizontal: Specifies lines that are spaced 50 percent closer than lines in the HatchStyleHorizontal pattern.

HatchStyleDashedDownwardDiagonal: Specifies dashed diagonal lines that slant to the right from top to bottom points.

HatchStyleDashedUpwardDiagonal: Specifies dashed diagonal lines that slant to the left from top to bottom points.

HatchStyleDashedHorizontal: Specifies dashed horizontal lines.

HatchStyleDashedVertical: Specifies dashed vertical lines.

HatchStyleSmallConfetti: Specifies a pattern of lines that has the appearance of confetti.

HatchStyleLargeConfetti: Specifies a pattern of lines that has the appearance of confetti, and is composed of larger pieces than the HatchStyleSmallConfetti pattern.

HatchStyleZigZag: Specifies horizontal lines that are composed of zigzags.

HatchStyleWave: Specifies horizontal lines that are composed of tildes.

HatchStyleDiagonalBrick: Specifies a pattern of lines that has the appearance of layered bricks that slant to the left from top to bottom points.

HatchStyleHorizontalBrick: Specifies a pattern of lines that has the appearance of horizontally layered bricks.

HatchStyleWeave: Specifies a pattern of lines that has the appearance of a woven material.

HatchStylePlaid: Specifies a pattern of lines that has the appearance of a plaid material.

HatchStyleDivot: Specifies a pattern of lines that has the appearance of divots.

HatchStyleDottedGrid: Specifies crossing horizontal and vertical lines, each of which is composed of dots.

HatchStyleDottedDiamond: Specifies crossing forward and backward diagonal lines, each of which is composed of dots.

HatchStyleShingle: Specifies a pattern of lines that has the appearance of diagonally layered shingles that slant to the right from top to bottom points.

HatchStyleTrellis: Specifies a pattern of lines that has the appearance of a trellis.

HatchStyleSphere: Specifies a pattern of lines that has the appearance of spheres laid adjacent to each other.

HatchStyleSmallGrid: Specifies crossing horizontal and vertical lines that are spaced 50 percent closer together than HatchStyleLargeGrid.

HatchStyleSmallCheckerBoard: Specifies a pattern of lines that has the appearance of a checkerboard.

HatchStyleLargeCheckerBoard: Specifies a pattern of lines that has the appearance of a checkerboard, with squares that are twice the size of the squares in the **HatchStyleSmallCheckerBoard** pattern.

HatchStyleOutlinedDiamond: Specifies crossing forward and backward diagonal lines; the lines are not anti-aliased.

HatchStyleSolidDiamond: Specifies a pattern of lines that has the appearance of a checkerboard placed diagonally.

Graphics brushes are specified by EmfPlusBrush objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.14 HotkeyPrefix Enumeration

The HotkeyPrefix enumeration defines output options for hotkey prefixes in graphics text.

```
typedef  enum
{
  HotkeyPrefixNone = 0x00000000,
  HotkeyPrefixShow = 0x00000001,
```

```
  HotkeyPrefixHide = 0x00000002
} HotkeyPrefix;
```

HotkeyPrefixNone: Specifies that the hotkey prefix SHOULD NOT be displayed.

HotkeyPrefixShow: Specifies that no hotkey prefix is defined.

HotkeyPrefixHide: Specifies that the hotkey prefix SHOULD be displayed.

Graphics text is specified by EmfPlusStringFormat objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.15 ImageDataType Enumeration

The ImageDataType enumeration defines types of image data formats.

```
typedef  enum
{
  ImageDataTypeUnknown = 0x00000000,
  ImageDataTypeBitmap = 0x00000001,
  ImageDataTypeMetafile = 0x00000002
} ImageDataType;
```

ImageDataTypeUnknown: The type of image is not known.

ImageDataTypeBitmap: Specifies a bitmap image.

ImageDataTypeMetafile: Specifies a metafile image.

Graphics images are specified by EmfPlusImage objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.16 InterpolationMode Enumeration

The InterpolationMode enumeration defines ways to perform scaling, including stretching and shrinking.

```
typedef  enum
{
  InterpolationModeDefault = 0x00,
  InterpolationModeLowQuality = 0x01,
  InterpolationModeHighQuality = 0x02,
  InterpolationModeBilinear = 0x03,
  InterpolationModeBicubic = 0x04,
  InterpolationModeNearestNeighbor = 0x05,
  InterpolationModeHighQualityBilinear = 0x06,
  InterpolationModeHighQualityBicubic = 0x07
} InterpolationMode;
```

InterpolationModeDefault: Specifies the default interpolation mode, which is defined as InterpolationModeBilinear.

InterpolationModeLowQuality: Specifies a low-quality interpolation mode, which is defined as InterpolationModeNearestNeighbor.

InterpolationModeHighQuality: Specifies a high-quality interpolation mode, which is defined as InterpolationModeHighQualityBicubic.

InterpolationModeBilinear: Specifies bilinear interpolation, which uses the closest 2x2 neighborhood of known pixels surrounding the interpolated pixel. The weighted average of these 4 known pixel values determines the value to assign to the interpolated pixel. The result is smoother looking than InterpolationModeNearestNeighbor.

InterpolationModeBicubic: Specifies bicubic interpolation, which uses the closest 4x4 neighborhood of known pixels surrounding the interpolated pixel. The weighted average of these 16 known pixel values determines the value to assign to the interpolated pixel. Because the known pixels are likely to be at varying distances from the interpolated pixel, closer pixels are given a higher weight in the calculation. The result is smoother looking than InterpolationModeBilinear.

InterpolationModeNearestNeighbor: Specifies nearest-neighbor interpolation, which uses only the value of the pixel that is closest to the interpolated pixel. This mode simply duplicates or removes pixels, producing the lowest-quality result among these options.

InterpolationModeHighQualityBilinear: Specifies bilinear interpolation with prefiltering.

InterpolationModeHighQualityBicubic: Specifies bicubic interpolation with prefiltering, which produces the highest-quality result among these options.

To stretch an image, each pixel in the original image SHOULD be mapped to a group of pixels in the larger image. To shrink an image, groups of pixels in the original image SHOULD be mapped to single pixels in the smaller image. The effectiveness of the algorithm that performs these mappings determines the quality of a scaled image. Higher-quality interpolation generally uses more data points and requires more processing time than lower-quality interpolation.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.17 LanguageIdentifier Enumeration

The LanguageIdentifier enumeration defines identifiers for natural languages in locales, including countries, geographical regions, and administrative districts.

```
typedef  enum
{
  LANG_NEUTRAL = 0x0000,
  zh-CHS = 0x0004,
  LANG_INVARIANT = 0x007F,
  LANG_NEUTRAL_USER_DEFAULT = 0x0400,
  ar-SA = 0x0401,
  bg-BG = 0x0402,
  ca-ES = 0x0403,
  zh-CHT = 0x0404,
  cs-CZ = 0x0405,
  da-DK = 0x0406,
  de-DE = 0x0407,
  el-GR = 0x0408,
  en-US = 0x0409,
  es-Tradnl-ES = 0x040A,
  fi-FI = 0x040B,
  fr-FR = 0x040C,
  he-IL = 0x040D,
  hu-HU = 0x040E,
  is-IS = 0x040F,
  it-IT = 0x0410,
  ja-JA = 0x0411,
  ko-KR = 0x0412,
  nl-NL = 0x0413,
  nb-NO = 0x0414,
  pl-PL = 0x0415,
```

```
pt-BR = 0x0416,
rm-CH = 0x0417,
ro-RO = 0x0418,
ru-RU = 0x0419,
hr-HR = 0x041A,
sk-SK = 0x041B,
sq-AL = 0x041C,
sv-SE = 0x041D,
th-TH = 0x041E,
tr-TR = 0x041F,
ur-PK = 0x0420,
id-ID = 0x0421,
uk-UA = 0x0422,
be-BY = 0x0423,
sl-SI = 0x0424,
et-EE = 0x0425,
lv-LV = 0x0426,
lt-LT = 0x0427,
tg-TJ = 0x0428,
fa-IR = 0x0429,
vi-VN = 0x042A,
hy-AM = 0x042B,
az-Latn-AZ = 0x042C,
eu-ES = 0x042D,
wen-DE = 0x042E,
mk-MK = 0x042F,
st-ZA = 0x0430,
tn-ZA = 0x0432,
xh-ZA = 0x0434,
zu-ZA = 0x0435,
af-ZA = 0x0436,
ka-GE = 0x0437,
fa-FA = 0x0438,
hi-IN = 0x0439,
mt-MT = 0x043A,
se-NO = 0x043B,
ga-GB = 0x043C,
ms-MY = 0x043E,
kk-KZ = 0x043F,
ky-KG = 0x0440,
sw-KE = 0x0441,
tk-TM = 0x0442,
uz-Latn-UZ = 0x0443,
tt-Ru = 0x0444,
bn-IN = 0x0445,
pa-IN = 0x0446,
gu-IN = 0x0447,
or-IN = 0x0448,
ta-IN = 0x0449,
te-IN = 0x044A,
kn-IN = 0x044B,
ml-IN = 0x044C,
as-IN = 0x044D,
mr-IN = 0x044E,
sa-IN = 0x044F,
mn-MN = 0x0450,
bo-CN = 0x0451,
cy-GB = 0x0452,
km-KH = 0x0453,
lo-LA = 0x0454,
gl-ES = 0x0456,
kok-IN = 0x0457,
sd-IN = 0x0459,
syr-SY = 0x045A,
si-LK = 0x045B,
iu-Cans-CA = 0x045D,
am-ET = 0x045E,
ne-NP = 0x0461,
fy-NL = 0x0462,
```

```
ps-AF = 0x0463,
fil-PH = 0x0464,
div-MV = 0x0465,
ha-Latn-NG = 0x0468,
yo-NG = 0x046A,
quz-BO = 0x046B,
nzo-ZA = 0x046C,
ba-RU = 0x046D,
lb-LU = 0x046E,
kl-GL = 0x046F,
ig-NG = 0x0470,
so-SO = 0x0477,
ii-CN = 0x0478,
arn-CL = 0x047A,
moh-CA = 0x047C,
br-FR = 0x047E,
ug-CN = 0x0480,
mi-NZ = 0x0481,
oc-FR = 0x0482,
co-FR = 0x0483,
gsw-FR = 0x0484,
sah-RU = 0x0485,
qut-GT = 0x0486,
rw-RW = 0x0487,
wo-SN = 0x0488,
gbz-AF = 0x048C,
LANG_NEUTRAL_SYS_DEFAULT = 0x0800,
ar-IQ = 0x0801,
zh-CN = 0x0804,
de-CH = 0x0807,
en-GB = 0x0809,
es-MX = 0x080A,
fr-BE = 0x080C,
it-CH = 0x0810,
ko-Johab-KR = 0x0812,
nl-BE = 0x0813,
nn-NO = 0x0814,
pt-PT = 0x0816,
sr-Latn-SP = 0x081A,
sv-FI = 0x081D,
ur-IN = 0x0820,
lt-C-LT = 0x0827,
az-Cyrl-AZ = 0x082C,
wee-DE = 0x082E,
se-SE = 0x083B,
ga-IE = 0x083C,
ms-BN = 0x083E,
uz-Cyrl-UZ = 0x0843,
bn-BD = 0x0845,
mn-Mong-CN = 0x0850,
sd-PK = 0x0859,
iu-Latn-CA = 0x085D,
tzm-Latn-DZ = 0x085F,
quz-EC = 0x086B,
LANG_NEUTRAL_CUSTOM_DEFAULT = 0x0C00,
ar-EG = 0x0C01,
zh-HK = 0x0C04,
de-AT = 0x0C07,
en-AU = 0x0C09,
es-ES = 0x0C0A,
fr-CA = 0x0C0C,
sr-Cyrl-CS = 0x0C1A,
se-FI = 0x0C3B,
quz-PE = 0x0C6B,
LANG NEUTRAL CUSTOM = 0x1000,
ar-LY = 0x1001,
zh-SG = 0x1004,
de-LU = 0x1007,
en-CA = 0x1009,
```

```
    es-GT = 0x100A,
    fr-CH = 0x100C,
    hr-BA = 0x101A,
    smj-NO = 0x103B,
    LANG_NEUTRAL_CUSTOM_DEFAULT_MUI = 0x1400,
    ar-DZ = 0x1401,
    zh-MO = 0x1404,
    de-LI = 0x1407,
    en-NZ = 0x1409,
    es-CR = 0x140A,
    fr-LU = 0x140C,
    bs-Latn-BA = 0x141A,
    smj-SE = 0x143B,
    ar-MA = 0x1801,
    en-IE = 0x1809,
    es-PA = 0x180A,
    ar-MC = 0x180C,
    sr-Latn-BA = 0x181A,
    sma-NO = 0x183B,
    ar-TN = 0x1C01,
    en-ZA = 0x1C09,
    es-DO = 0x1C0A,
    sr-Cyrl-BA = 0x1C1A,
    sma-SE = 0x1C3B,
    ar-OM = 0x2001,
    el-2-GR = 0x2008,
    en-JM = 0x2009,
    es-VE = 0x200A,
    bs-Cyrl-BA = 0x201A,
    sms-FI = 0x203B,
    ar-YE = 0x2401,
    ar-029 = 0x2409,
    es-CO = 0x240A,
    smn-FI = 0x243B,
    ar-SY = 0x2801,
    en-BZ = 0x2809,
    es-PE = 0x280A,
    ar-JO = 0x2C01,
    en-TT = 0x2C09,
    es-AR = 0x2C0A,
    ar-LB = 0x3001,
    en-ZW = 0x3009,
    es-EC = 0x300A,
    ar-KW = 0x3401,
    en-PH = 0x3409,
    es-CL = 0x340A,
    ar-AE = 0x3801,
    es-UY = 0x380A,
    ar-BH = 0x3C01,
    es-PY = 0x3C0A,
    ar-QA = 0x4001,
    en-IN = 0x4009,
    es-BO = 0x400A,
    en-MY = 0x4409,
    es-SV = 0x440A,
    en-SG = 0x4809,
    es-HN = 0x480A,
    es-NI = 0x4C0A,
    es-PR = 0x500A,
    es-US = 0x540A,
    zh-Hant = 0x7C04
  } LanguageIdentifiers;
```

LANG_NEUTRAL: Neutral locale language.

zh-CHS: Chinese, Simplified (China).

LANG_INVARIANT: Invariant language.

LANG_NEUTRAL_USER_DEFAULT: User default locale language.

ar-SA: Arabic (Saudi Arabia).

bg-BG: Bulgarian (Bulgaria).

ca-ES: Catalan (Spain).

zh-CHT: Chinese, Traditional (Taiwan).

cs-CZ: Czech (Czech Republic).

da-DK: Danish (Denmark).

de-DE: German (Germany).

el-GR: Greek (Greece).

en-US: English (United States).

es-Tradnl-ES: Spanish, Traditional (Spain).

fi-FI: Finnish (Finland).

fr-FR: French (France).

he-IL: Hebrew (Israel).

hu-HU: Hungarian (Hungary).

is-IS: Icelandic (Iceland).

it-IT: Italian (Italy).

ja-JA: Japanese (Japan).

ko-KR: Korean (Korea).

nl-NL: Dutch (Netherlands).

nb-NO: Bokmal (Norway).

pl-PL: Polish (Poland).

pt-BR: Portuguese (Brazil).

rm-CH: Romansh (Switzerland).

ro-RO: Romanian (Romania).

ru-RU: Russian (Russia).

hr-HR: Croatian (Croatia).

sk-SK: Slovak (Slovakia).

sq-AL: Albanian (Albania).

sv-SE: Swedish (Sweden).

th-TH: Thai (Thailand).

tr-TR: Turkish (Turkey).

ur-PK: Urdu (Pakistan).

id-ID: Indonesian (Indonesia).

uk-UA: Ukranian (Ukraine).

be-BY: Belarusian (Belarus).

sl-SI: Slovenian (Slovenia).

et-EE: Estonian (Estonia).

lv-LV: Latvian (Latvia).

lt-LT: Lithuanian (Lithuania).

tg-TJ: Tajik (Tajikistan).

fa-IR: Persian (Iran).

vi-VN: Vietnamese (Vietnam).

hy-AM: Armenian (Armenia).

az-Latn-AZ: Azeri, Latin alphabet (Azerbaijan).

eu-ES: Basque (Spain).

wen-DE: Sorbian, Upper (Germany).

mk-MK: Macedonian (Macedonia).

st-ZA: Sutu (South Africa).

tn-ZA: Setswana (Botswana).

xh-ZA: isiXhosa (South Africa).

zu-ZA: isiZulu (South Africa).

af-ZA: Afrikaans (South Africa).

ka-GE: Georgian (Georgia).

fa-FA: Faeroese (Faroe Islands).

hi-IN: Hindi (India).

mt-MT: Maltese (Malta).

se-NO: Sami, Northern (Norway).

ga-GB: Gaelic (United Kingdom).

ms-MY: Malay (Malaysia).

kk-KZ: Kazakh (Kazakhstan).

ky-KG: Kyrgyz (Kyrgyzstan).

sw-KE: Kiswahili (Kenya, Tanzania, and other Eastern African nations; and it is the official language of the African Union).

tk-TM: Turkmen (Turkmenistan).

uz-Latn-UZ: Uzbek, Latin alphabet (Uzbekistan).

tt-Ru: Tatar (Belarus, Russia, Ukraine, and other eastern European nations; and Kazakhstan, and Uzbekistan in central Asia).

bn-IN: Bengali, Bengali script (India).

pa-IN: Punjabi (India).

gu-IN: Gujarati (India).

or-IN: Oriya (India).

ta-IN: Tamil (India, Sri Lanka).

te-IN: Telugu (India).

kn-IN: Kannada (India).

ml-IN: Malayalam (India).

as-IN: Assamese (India).

mr-IN: Marathi (India).

sa-IN: Sanskrit (India).

mn-MN: Mongolian, Cyrillic alphabet (Mongolia).

bo-CN: Tibetan (China).

cy-GB: Welsh (United Kingdom).

km-KH: Khmer (Cambodia).

lo-LA: Lao (Laos).

gl-ES: Galician (Spain).

kok-IN: Konkani (India).

sd-IN: Sindhi (India).

syr-SY: Syriac (Syria).

si-LK: Sinhalese (Sri Lanka).

iu-Cans-CA: Inuktitut, Syllabics (Canada).

am-ET: Amharic (Ethiopia).

ne-NP: Nepali (Nepal).

fy-NL: Frisian (Netherlands).

ps-AF: Pashto (Afghanistan, Pakistan).

fil-PH: Filipino (Philippines).

div-MV: Divehi (Maldives, India).

ha-Latn-NG: Hausa, Latin alphabet (Benin, Nigeria, Togo, and other western African nations).

yo-NG: Yoruba (Benin, Ghana, Nigeria, Togo, and other western African nations).

quz-BO: Quechua (Bolivia).

nzo-ZA: Sesotho sa Leboa (South Africa).

ba-RU: Bashkir (Russia).

lb-LU: Luxembourgish (Luxembourg).

kl-GL: Greenlandic (Greenland).

ig-NG: Igbo (Nigeria).

so-SO: Somali (Somalia).

ii-CN: Yi (China).

arn-CL: Mapudungun (Chile).

moh-CA: Mohawk (Canada).

br-FR: Breton (France).

ug-CN: Uighur (China).

mi-NZ: Maori (New Zealand).

oc-FR: Occitan (France).

co-FR: Corsican (France).

gsw-FR: Alsatian (France).

sah-RU: Yakut (Russia).

qut-GT: K'iche (Guatemala).

rw-RW: Kinyarwanda (Rwanda).

wo-SN: Wolof (Gambia, Mauritania, Senegal, and other western African nations).

gbz-AF: Dari (Afghanistan).

LANG_NEUTRAL_SYS_DEFAULT: System default locale language.

ar-IQ: Arabic (Iraq).

zh-CN: Chinese (China).

de-CH: German (Switzerland).

en-GB: English (United Kingdom).

es-MX: Spanish (Mexico).

fr-BE: French (Belgium).

it-CH: Italian (Switzerland).

ko-Johab-KR: Korean, Johab (Korea).

nl-BE: Dutch (Belgium).

nn-NO: Nyorsk (Norway).

pt-PT: Portuguese (Portugal).

sr-Latn-SP: Serbian, Latin alphabet (Serbia).

sv-FI: Swedish (Finland).

ur-IN: Urdu (India).

lt-C-LT: Lithuanian, Classic (Lithuania).

az-Cyrl-AZ: Azeri, Cyrillic alphabet (Azerbaijan).

wee-DE: Sorbian, Lower (Germany).

se-SE: Sami, Northern (Sweden).

ga-IE: Irish (Ireland).

ms-BN: Malay (Brunei).

uz-Cyrl-UZ: Uzbek, Cyrillic alphabet (Uzbekistan).

bn-BD: Bengali (Bangladesh).

mn-Mong-CN: Mongolian, Traditional (China).

sd-PK: Sindhi (Pakistan).

iu-Latn-CA: Inuktitut, Latin alphabet (Canada).

tzm-Latn-DZ: Tamazight, Latin alphabet (Algeria).

quz-EC: Quechua (Ecuador).

LANG_NEUTRAL_CUSTOM_DEFAULT: Default custom locale language.

ar-EG: Arabic (Egypt).

zh-HK: Chinese (Hong Kong Special Administrative Region, China).

de-AT: German (Austria).

en-AU: English (Australia).

es-ES: Spanish, Modern (Spain).

fr-CA: French (Canada).

sr-Cyrl-CS: Serbian, Cyrillic alphabet (Serbia).

se-FI: Sami, Northern (Finland).

quz-PE: Quechua (Peru).

LANG_NEUTRAL_CUSTOM: Unspecified custom locale language.

ar-LY: Arabic (Libya).

zh-SG: Chinese (Singapore).

de-LU: German (Luxembourg).

en-CA: English (Canada).

es-GT: Spanish (Guatemala).

fr-CH: French (Switzerland).

hr-BA: Croatian (Bosnia and Herzegovina).

smj-NO: Sami, Luli (Norway).

LANG_NEUTRAL_CUSTOM_DEFAULT_MUI: Default custom multi-user interface locale language.

ar-DZ: Arabic (Algeria).

zh-MO: Chinese (Macao Special Administrative Region, China).

de-LI: German (Liechtenstein).

en-NZ: English (New Zealand).

es-CR: Spanish (Costa Rica).

fr-LU: French (Luxembourg).

bs-Latn-BA: Bosnian, Latin alphabet (Bosnia and Herzegovina).

smj-SE: Sami, Lule (Sweden).

ar-MA: Arabic (Morocco).

en-IE: English (Ireland).

es-PA: Spanish (Panama).

ar-MC: French (Monaco).

sr-Latn-BA: Serbian, Latin alphabet (Bosnia and Herzegovina).

sma-NO: Sami, Southern (Norway).

ar-TN: Arabic (Tunisia).

en-ZA: English (South Africa).

es-DO: Spanish (Dominican Republic).

sr-Cyrl-BA: Serbian, Cyrillic alphabet (Bosnia and Herzegovina).

sma-SE: Sami, Southern (Sweden).

ar-OM: Arabic (Oman).

el-2-GR: Greek 2 (Greece).

en-JM: English (Jamaica).

es-VE: Spanish (Venezuela).

bs-Cyrl-BA: Bosnian, Cyrillic alphabet (Bosnia and Herzegovina).

sms-FI: Sami, Skolt (Finland).

ar-YE: Arabic (Yemen).

ar-029: English (Nations of the Caribbean).

es-CO: Spanish (Colombia).

smn-FI: Sami, Inari (Finland).

ar-SY: Arabic (Syria).

en-BZ: English (Belize).

es-PE: Spanish (Peru).

ar-JO: Arabic (Jordan).

en-TT: English (Trinidad and Tobago).

es-AR: Spanish (Argentina).

ar-LB: Arabic (Lebanon).

en-ZW: English (Zimbabwe).

es-EC: Spanish (Ecuador).

ar-KW: Arabic (Kuwait).

en-PH: English (Phillippines).

es-CL: Spanish (Chile).

ar-AE: Arabic (United Arab Emirates).

es-UY: Spanish (Uruguay).

ar-BH: Arabic (Bahrain).

es-PY: Spanish (Paraguay).

ar-QA: Arabic (Qatar).

en-IN: English (India).

es-BO: Spanish (Bolivia).

en-MY: English (Malaysia).

es-SV: Spanish (El Salvador).

en-SG: English (Singapore).

es-HN: Spanish (Honduras).

es-NI: Spanish (Nicaragua).

es-PR: Spanish (Puerto Rico).

es-US: Spanish (United States).

zh-Hant: Chinese, Traditional (China).

Language identifiers are specified by EmfPlusLanguageIdentifier objects. Each value is an encoding of a primary language identifier and a sublanguage identifier.

See [MS-LCID] for additional information concerning language identifiers, including Windows version support.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.18 LineCapType Enumeration

The LineCapType enumeration defines types of line caps to use at the ends of lines that are drawn with graphics pens.

```
typedef  enum
{
  LineCapTypeFlat = 0x00000000,
  LineCapTypeSquare = 0x00000001,
  LineCapTypeRound = 0x00000002,
  LineCapTypeTriangle = 0x00000003,
  LineCapTypeNoAnchor = 0x00000010,
  LineCapTypeSquareAnchor = 0x00000011,
  LineCapTypeRoundAnchor = 0x00000012,
  LineCapTypeDiamondAnchor = 0x00000013,
  LineCapTypeArrowAnchor = 0x00000014,
  LineCapTypeAnchorMask = 0x000000F0,
  LineCapTypeCustom = 0x000000FF
} LineCapType;
```

LineCapTypeFlat: Specifies a squared-off line cap. The end of the line MUST be the last point in the line.

LineCapTypeSquare: Specifies a square line cap. The center of the square MUST be located at the last point in the line. The width of the square is the line width.

LineCapTypeRound: Specifies a circular line cap. The center of the circle MUST be located at the last point in the line. The diameter of the circle is the line width.

LineCapTypeTriangle: Specifies a triangular line cap. The base of the triangle MUST be located at the last point in the line. The base of the triangle is the line width.

LineCapTypeNoAnchor: Specifies that the line end is not anchored.

LineCapTypeSquareAnchor: Specifies that the line end is anchored with a square line cap. The center of the square MUST be located at the last point in the line. The height and width of the square are the line width.

LineCapTypeRoundAnchor: Specifies that the line end is anchored with a circular line cap. The center of the circle MUST be located at the last point in the line. The circle SHOULD be wider than the line.

LineCapTypeDiamondAnchor: Specifies that the line end is anchored with a diamond-shaped line cap, which is a square turned at 45 degrees. The center of the diamond MUST be located at the last point in the line. The diamond SHOULD be wider than the line.

LineCapTypeArrowAnchor: Specifies that the line end is anchored with an arrowhead shape. The arrowhead point MUST be located at the last point in the line. The arrowhead SHOULD be wider than the line.

LineCapTypeAnchorMask: Mask used to check whether a line cap is an anchor cap.

LineCapTypeCustom: Specifies a custom line cap.

Graphics line caps are specified by EmfPlusPen objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.19 LineJoinType Enumeration

The LineJoinType enumeration defines ways to join two lines that are drawn by the same graphics pen and whose ends meet.

```
typedef  enum
{
  LineJoinTypeMiter = 0x00000000,
  LineJoinTypeBevel = 0x00000001,
  LineJoinTypeRound = 0x00000002,
  LineJoinTypeMiterClipped = 0x00000003
} LineJoinType;
```

LineJoinTypeMiter: Specifies a mitered line join.

LineJoinTypeBevel: Specifies a beveled line join.

LineJoinTypeRound: Specifies a rounded line join.

LineJoinTypeMiterClipped: Specifies a clipped mitered line join.

Graphics lines are specified by EmfPlusPen objects. A line join makes the intersection of the two line ends look more continuous.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.20 LineStyle Enumeration

The LineStyle enumeration defines styles of lines that are drawn with graphics pens.

```
typedef  enum
{
  LineStyleSolid = 0x00000000,
  LineStyleDash = 0x00000001,
  LineStyleDot = 0x00000002,
  LineStyleDashDot = 0x00000003,
  LineStyleDashDotDot = 0x00000004,
  LineStyleCustom = 0x00000005
} LineStyle;
```

LineStyleSolid: Specifies a solid line.

LineStyleDash: Specifies a dashed line.

LineStyleDot: Specifies a dotted line.

LineStyleDashDot: Specifies an alternating dash-dot line.

LineStyleDashDotDot: Specifies an alternating dash-dot-dot line.

LineStyleCustom: Specifies a user-defined, custom dashed line.

Graphics lines are specified by EmfPlusPen objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.21 MetafileDataType Enumeration

The MetafileDataType enumeration defines types of metafiles data that can be embedded in an EMF+ metafile.

```
typedef  enum
{
  MetafileDataTypeWmf = 0x00000001,
  MetafileDataTypeWmfPlaceable = 0x00000002,
  MetafileDataTypeEmf = 0x00000003,
  MetafileDataTypeEmfPlusOnly = 0x00000004,
  MetafileDataTypeEmfPlusDual = 0x00000005
} MetafileDataType;
```

MetafileDataTypeWmf: Specifies that the metafile is a WMF metafile [MS-WMF] that specifies graphics operations with WMF records.

MetafileDataTypeWmfPlaceable: Specifies that the metafile is a WMF metafile that specifies graphics operations with WMF records, and which contains additional header information that makes the WMF metafile device-independent.

MetafileDataTypeEmf: Specifies that the metafile is an EMF metafile that specifies graphics operations with EMF records ([MS-EMF] section 2.3).

MetafileDataTypeEmfPlusOnly: Specifies that the metafile is an EMF+ metafile that specifies graphics operations with EMF+ records only.

MetafileDataTypeEmfPlusDual: Specifies that the metafile is an EMF+ metafile that specifies graphics operations with both EMF and EMF+ records.

Embedded metafile data is specified by EmfPlusMetafileData objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.22 ObjectType Enumeration

The ObjectType enumeration defines types of graphics objects that can be created and used in graphics operations.

```
typedef  enum
{
  ObjectTypeInvalid = 0x00000000,
  ObjectTypeBrush = 0x00000001,
  ObjectTypePen = 0x00000002,
  ObjectTypePath = 0x00000003,
  ObjectTypeRegion = 0x00000004,
  ObjectTypeImage = 0x00000005,
  ObjectTypeFont = 0x00000006,
  ObjectTypeStringFormat = 0x00000007,
  ObjectTypeImageAttributes = 0x00000008,
  ObjectTypeCustomLineCap = 0x00000009
} ObjectType;
```

ObjectTypeInvalid: The object is not a valid object.

ObjectTypeBrush: Specifies an EmfPlusBrush object. Brush objects fill graphics regions.

ObjectTypePen: Specifies an EmfPlusPen object. Pen objects draw graphics lines.

ObjectTypePath: Specifies an EmfPlusPath object. Path objects specify sequences of lines, curves, and shapes.

ObjectTypeRegion: Specifies an EmfPlusRegion object. Region objects specify areas of the output surface.

ObjectTypeImage: Specifies an EmfPlusImage object. Image objects encapsulate bitmaps and metafiles.

ObjectTypeFont: Specifies an EmfPlusFont object. Font objects specify font properties, including **typeface** style, **em size**, and **font family**.

ObjectTypeStringFormat: Specifies an EmfPlusStringFormat object. String format objects specify text layout, including alignment, orientation, tab stops, clipping, and digit substitution for languages that do not use Western European digits.

ObjectTypeImageAttributes: Specifies an EmfPlusImageAttributes object. Image attribute objects specify operations on pixels during image rendering, including color adjustment, **grayscale** adjustment, gamma correction, and **color mapping**.

ObjectTypeCustomLineCap: Specifies an EmfPlusCustomLineCap object. Custom line cap objects specify shapes to draw at the ends of a graphics line, including squares, circles, and diamonds.

Graphics objects are specified by EmfPlusObject records.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.23 PathPointType Enumeration

The PathPointType enumeration defines types of points on a graphics path.

```
typedef  enum
{
  PathPointTypeStart = 0x00,
  PathPointTypeLine = 0x01,
  PathPointTypeBezier = 0x03
} PathPointType;
```

PathPointTypeStart: Specifies that the point is the starting point of a path.

PathPointTypeLine: Specifies that the point is one of the two endpoints of a line.

PathPointTypeBezier: Specifies that the point is an endpoint or control point of a cubic Bezier curve.

Graphics path point types are specified by EmfPlusPathPointType objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.24 PenAlignment Enumeration

The PenAlignment enumeration defines the distribution of the width of the pen with respect to the line being drawn.

```
typedef  enum
{
  PenAlignmentCenter = 0x00000000,
  PenAlignmentInset = 0x00000001,
  PenAlignmentLeft = 0x00000002,
```

```
  PenAlignmentOutset = 0x00000003,
  PenAlignmentRight = 0x00000004
} PenAlignment;
```

PenAlignmentCenter: Specifies that the EmfPlusPen object is centered over the theoretical line.

PenAlignmentInset: Specifies that the pen is positioned on the inside of the theoretical line.

PenAlignmentLeft: Specifies that the pen is positioned to the left of the theoretical line.

PenAlignmentOutset: Specifies that the pen is positioned on the outside of the theoretical line.

PenAlignmentRight: Specifies that the pen is positioned to the right of the theoretical line.

Graphics pens are specified by EmfPlusPen objects. Pen alignment can be visualized by considering a theoretical one-dimensional line drawn between two specified points. The pen alignment determines the proportion of pen width that is orthogonal to the theoretical line.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.25 PixelFormat Enumeration

The PixelFormat enumeration defines pixel formats that are supported in EMF+ bitmaps.

```
typedef  enum
{
  PixelFormatUndefined = 0x00000000,
  PixelFormat1bppIndexed = 0x00030101,
  PixelFormat4bppIndexed = 0x00030402,
  PixelFormat8bppIndexed = 0x00030803,
  PixelFormat16bppGrayScale = 0x00101004,
  PixelFormat16bppRGB555 = 0x00021005,
  PixelFormat16bppRGB565 = 0x00021006,
  PixelFormat16bppARGB1555 = 0x00061007,
  PixelFormat24bppRGB = 0x00021808,
  PixelFormat32bppRGB = 0x00022009,
  PixelFormat32bppARGB = 0x0026200A,
  PixelFormat32bppPARGB = 0x000E200B,
  PixelFormat48bppRGB = 0x0010300C,
  PixelFormat64bppARGB = 0x0034400D,
  PixelFormat64bppPARGB = 0x001A400E
} PixelFormat;
```

PixelFormatUndefined: The format is not specified.

PixelFormat1bppIndexed: The format is monochrome, and a color palette lookup table is used.

PixelFormat4bppIndexed: The format is 16-color, and a color palette lookup table is used.

PixelFormat8bppIndexed: The format is 256-color, and a color palette lookup table is used.

PixelFormat16bppGrayScale: The format is 16 bits per pixel, grayscale.

PixelFormat16bppRGB555: The format is 16 bits per pixel; 5 bits each are used for the red, green, and blue components. The remaining bit is not used.

PixelFormat16bppRGB565: The format is 16 bits per pixel; 5 bits are used for the red component, 6 bits for the green component, and 5 bits for the blue component.

PixelFormat16bppARGB1555: The format is 16 bits per pixel; 1 bit is used for the alpha component, and 5 bits each are used for the red, green, and blue components.

PixelFormat24bppRGB: The format is 24 bits per pixel; 8 bits each are used for the red, green, and blue components.

PixelFormat32bppRGB: The format is 32 bits per pixel; 8 bits each are used for the red, green, and blue components. The remaining 8 bits are not used.

PixelFormat32bppARGB: The format is 32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components.

PixelFormat32bppPARGB: The format is 32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components. The red, green, and blue components are premultiplied according to the alpha component.

PixelFormat48bppRGB: The format is 48 bits per pixel; 16 bits each are used for the red, green, and blue components.

PixelFormat64bppARGB: The format is 64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components.

PixelFormat64bppPARGB: The format is 64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components. The red, green, and blue components are premultiplied according to the alpha component.

Pixel formats are specified by EmfPlusBitmap objects. They are encoded as follows:

- Bits 0-7: Enumeration of the pixel format constants, starting at zero.
- Bits 8-15: The total number of bits per pixel.
- Bit 16: If set, the color value is indexed into a palette.
- Bit 17: If set, the color value is in a **GDI**-supported format.
- Bit 18: If set, the color value has an alpha component.
- Bit 19: If set, the color value has a premultiplied alpha component.
- Bit 20: If set, extended colors, 16-bits per channel, are supported.
- Bits 21-31: Reserved.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.26 PixelOffsetMode Enumeration

The PixelOffsetMode enumeration defines how pixels are offset, which specifies the trade-off between rendering speed and quality.

```
typedef  enum
{
  PixelOffsetModeDefault = 0x00,
  PixelOffsetModeHighSpeed = 0x01,
  PixelOffsetModeHighQuality = 0x02,
  PixelOffsetModeNone = 0x03,
  PixelOffsetModeHalf = 0x04
} PixelOffsetMode;
```

PixelOffsetModeDefault: Pixels are centered on integer coordinates, specifying speed over quality.

PixelOffsetModeHighSpeed: Pixels are centered on integer coordinates, as with PixelOffsetModeNone. Higher speed at the expense of quality is specified.

PixelOffsetModeHighQuality: Pixels are centered on half-integer coordinates, as with PixelOffsetModeHalf. Higher quality at the expense of speed is specified.

PixelOffsetModeNone: Pixels are centered on the origin, which means that the pixel covers the area from -0.5 to 0.5 on both the x and y axes and its center is at (0,0).

PixelOffsetModeHalf: Pixels are centered on half-integer coordinates, which means that the pixel covers the area from 0 to 1 on both the x and y axes and its center is at (0.5,0.5). By offsetting pixels during rendering, the render quality can be improved at the cost of render speed.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.27 RegionNodeDataType Enumeration

The RegionNodeDataType enumeration defines types of region node data.

```
typedef  enum
{
  RegionNodeDataTypeAnd = 0x00000001,
  RegionNodeDataTypeOr = 0x00000002,
  RegionNodeDataTypeXor = 0x00000003,
  RegionNodeDataTypeExclude = 0x00000004,
  RegionNodeDataTypeComplement = 0x00000005,
  RegionNodeDataTypeRect = 0x10000000,
  RegionNodeDataTypePath = 0x10000001,
  RegionNodeDataTypeEmpty = 0x10000002,
  RegionNodeDataTypeInfinite = 0x10000003
} RegionNodeDataType;
```

RegionNodeDataTypeAnd: Specifies a region node with child nodes. A Boolean AND operation SHOULD be applied to the left and right child nodes specified by an EmfPlusRegionNodeChildNodes object.

RegionNodeDataTypeOr: Specifies a region node with child nodes. A Boolean OR operation SHOULD be applied to the left and right child nodes specified by an EmfPlusRegionNodeChildNodes object.

RegionNodeDataTypeXor: Specifies a region node with child nodes. A Boolean XOR operation SHOULD be applied to the left and right child nodes specified by an EmfPlusRegionNodeChildNodes object.

RegionNodeDataTypeExclude: Specifies a region node with child nodes. A Boolean operation, defined as "the part of region 1 that is excluded from region 2", SHOULD be applied to the left and right child nodes specified by an EmfPlusRegionNodeChildNodes object.

RegionNodeDataTypeComplement: Specifies a region node with child nodes. A Boolean operation, defined as "the part of region 2 that is excluded from region 1", SHOULD be applied to the left and right child nodes specified by an EmfPlusRegionNodeChildNodes object.

RegionNodeDataTypeRect: Specifies a region node with no child nodes. The **RegionNodeData** field SHOULD specify a boundary with an EmfPlusRectF object.

RegionNodeDataTypePath: Specifies a region node with no child nodes. The **RegionNodeData** field SHOULD specify a boundary with an EmfPlusRegionNodePath object.

RegionNodeDataTypeEmpty: Specifies a region node with no child nodes. The **RegionNodeData** field SHOULD NOT be present.

RegionNodeDataTypeInfinite: Specifies a region node with no child nodes, and its bounds are not defined.

Region node data is specified by EmfPlusRegionNode objects.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.28 SmoothingMode Enumeration

The SmoothingMode enumeration defines smoothing modes to apply to lines, curves, and the edges of filled areas to make them appear more continuous or sharply defined.

```
typedef  enum
{
  SmoothingModeDefault = 0x00,
  SmoothingModeHighSpeed = 0x01,
  SmoothingModeHighQuality = 0x02,
  SmoothingModeNone = 0x03,
  SmoothingModeAntiAlias8x4 = 0x04,
  SmoothingModeAntiAlias8x8 = 0x05
} SmoothingMode;
```

SmoothingModeDefault: Default curve smoothing with no anti-aliasing.

SmoothingModeHighSpeed: Best performance with no anti-aliasing.

SmoothingModeHighQuality: Best quality with anti-aliasing.

SmoothingModeNone: No curve smoothing and no anti-aliasing.

SmoothingModeAntiAlias8x4: Anti-aliasing using an 8x4 box filter.<4>

SmoothingModeAntiAlias8x8: Anti-aliasing using an 8x8 box filter.<5>

See section 2.1.1 for the specification of additional enumerations.

2.1.1.29 StringAlignment Enumeration

The StringAlignment enumeration defines ways to align strings with respect to a text layout rectangle.

```
typedef  enum
{
  StringAlignmentNear = 0x00000000,
  StringAlignmentCenter = 0x00000001,
  StringAlignmentFar = 0x00000002
} StringAlignment;
```

StringAlignmentNear: Specifies that string alignment is toward the origin of the layout rectangle. This can be used to align characters along a line or to align text within a rectangle. For a right-to-left layout rectangle, the origin SHOULD be at the upper right.

StringAlignmentCenter: Specifies that alignment is centered between the origin and extent of the layout rectangle.

StringAlignmentFar: Specifies that alignment is to the right side of the layout rectangle.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.30 StringDigitSubstitution Enumeration

The StringDigitSubstitution enumeration defines ways to substitute digits in a string according to a user's locale or language.

```
typedef  enum
{
  StringDigitSubstitutionUser = 0x00000000,
  StringDigitSubstitutionNone = 0x00000001,
  StringDigitSubstitutionNational = 0x00000002,
  StringDigitSubstitutionTraditional = 0x00000003
} StringDigitSubstitution;
```

StringDigitSubstitutionUser: Specifies an implementation-defined substitution scheme.

StringDigitSubstitutionNone: Specifies to disable substitutions.

StringDigitSubstitutionNational: Specifies substitution digits that correspond with the official national language of the user's locale.

StringDigitSubstitutionTraditional: Specifies substitution digits that correspond to the user's native script or language, which can be different from the official national language of the user's locale.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.31 StringTrimming Enumeration

The StringTrimming enumeration defines how to trim characters from a string that is too large for the text layout rectangle.

```
typedef  enum
{
  StringTrimmingNone = 0x00000000,
  StringTrimmingCharacter = 0x00000001,
  StringTrimmingWord = 0x00000002,
  StringTrimmingEllipsisCharacter = 0x00000003,
  StringTrimmingEllipsisWord = 0x00000004,
  StringTrimmingEllipsisPath = 0x00000005
} StringTrimming;
```

StringTrimmingNone: Specifies that no trimming is done.

StringTrimmingCharacter: Specifies that the string is broken at the boundary of the last character that is inside the layout rectangle. This is the default.

StringTrimmingWord: Specifies that the string is broken at the boundary of the last word that is inside the layout rectangle.

StringTrimmingEllipsisCharacter: Specifies that the string is broken at the boundary of the last character that is inside the layout rectangle, and an ellipsis (...) is inserted after the character.

StringTrimmingEllipsisWord: Specifies that the string is broken at the boundary of the last word that is inside the layout rectangle, and an ellipsis (...) is inserted after the word.

StringTrimmingEllipsisPath: Specifies that the center is removed from the string and replaced by an ellipsis. The algorithm keeps as much of the last portion of the string as possible.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.32 TextRenderingHint Enumeration

The TextRenderingHint enumeration defines types of text hinting and anti-aliasing, which affects the quality of text rendering.

```
typedef  enum
{
  TextRenderingHintSystemDefault = 0x00,
  TextRenderingHintSingleBitPerPixelGridFit = 0x01,
  TextRenderingHintSingleBitPerPixel = 0x02,
  TextRenderingHintAntialiasGridFit = 0x03,
  TextRenderingHintAntialias = 0x04,
  TextRenderingHintClearTypeGridFit = 0x05
} TextRenderingHint;
```

TextRenderingHintSystemDefault: Specifies that each text character SHOULD be drawn using whatever font-smoothing settings have been configured on the operating system.

TextRenderingHintSingleBitPerPixelGridFit: Specifies that each text character SHOULD be drawn using its glyph bitmap. Smoothing MAY be used to improve the appearance of character glyph stems and curvature.

TextRenderingHintSingleBitPerPixel: Specifies that each text character SHOULD be drawn using its glyph bitmap. Smoothing is not used.

TextRenderingHintAntialiasGridFit: Specifies that each text character SHOULD be drawn using its anti-aliased glyph bitmap with smoothing. The rendering is high quality because of anti-aliasing, but at a higher performance cost.

TextRenderingHintAntialias: Specifies that each text character is drawn using its anti-aliased glyph bitmap without hinting. Better quality results from anti-aliasing, but stem width differences MAY be noticeable because hinting is turned off.

TextRenderingHintClearTypeGridFit: Specifies that each text character SHOULD be drawn using its **ClearType** glyph bitmap with smoothing. This is the highest-quality text hinting setting, which is used to take advantage of ClearType font features.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.33 UnitType Enumeration

The UnitType enumeration defines units of measurement in different coordinate systems.

```
typedef  enum
{
  UnitTypeWorld = 0x00,
  UnitTypeDisplay = 0x01,
  UnitTypePixel = 0x02,
  UnitTypePoint = 0x03,
  UnitTypeInch = 0x04,
  UnitTypeDocument = 0x05,
  UnitTypeMillimeter = 0x06
} UnitType;
```

UnitTypeWorld: Specifies a unit of logical distance within the world space.

UnitTypeDisplay: Specifies a unit of distance based on the characteristics of the physical display.

UnitTypePixel: Specifies a unit of 1 pixel.

UnitTypePoint: Specifies a unit of 1 printer's point, or 1/72 inch.

UnitTypeInch: Specifies a unit of 1 inch.

UnitTypeDocument: Specifies a unit of 1/300 inch.

UnitTypeMillimeter: Specifies a unit of 1 millimeter.

See section 2.1.1 for the specification of additional enumerations.

2.1.1.34 WrapMode Enumeration

The WrapMode enumeration defines how the pattern from a texture or gradient brush is tiled across a shape or at shape boundaries, when it is smaller than the area being filled.

```
typedef  enum
{
  WrapModeTile = 0x00000000,
  WrapModeTileFlipX = 0x00000001,
  WrapModeTileFlipY = 0x00000002,
  WrapModeTileFlipXY = 0x00000003,
  WrapModeClamp = 0x00000004
} WrapMode;
```

WrapModeTile: Tiles the gradient or texture.

WrapModeTileFlipX: Reverses the texture or gradient horizontally, and then tiles the texture or gradient.

WrapModeTileFlipY: Reverses the texture or gradient vertically, and then tiles the texture or gradient.

WrapModeTileFlipXY: Reverses the texture or gradient horizontally and vertically, and then tiles the texture or gradient.

WrapModeClamp: Fixes the texture or gradient to the object boundary.

See section 2.1.1 for the specification of additional enumerations.

2.1.2 Bit Flag Constant Types

The Bit Flag Constants specify properties and options for EMF+ objects and records. In general, bit flags can be combined with Boolean OR operations.

The following types of bit flags are defined:

Name	Section	Description
BrushData	2.1.2.1	Specifies properties of graphics brushes.
CustomLineCapData	2.1.2.2	Specifies data for custom line caps.
DriverStringOptions	2.1.2.3	Specifies properties of graphics text positioning and rendering.
FontStyle	2.1.2.4	Specifies styles of graphics font typefaces.
PaletteStyle	2.1.2.5	Specifies properties of graphics **palettes**.
PenData	2.1.2.7	Specifies properties of graphics pens.

Name	Section	Description
StringFormat	2.1.2.8	Specifies options for graphics text layout.

2.1.2.1 BrushData Flags

The BrushData flags specify properties of graphics brushes, including the presence of optional data fields. These flags can be combined to specify multiple options.

Constant/value	Description
BrushDataPath 0x00000001	This flag is meaningful in EmfPlusPathGradientBrushData objects. If set, an EmfPlusBoundaryPathData object MUST be specified in the **BoundaryData** field of the brush data object. If clear, an EmfPlusBoundaryPointData object MUST be specified in the **BoundaryData** field of the brush data object.
BrushDataTransform 0x00000002	This flag is meaningful in EmfPlusLinearGradientBrushData objects, EmfPlusPathGradientBrushData objects, and EmfPlusTextureBrushData objects. If set, a 2x3 world space to **device space** transform matrix MUST be specified in the **OptionalData** field of the brush data object.
BrushDataPresetColors 0x00000004	This flag is meaningful in EmfPlusLinearGradientBrushData and EmfPlusPathGradientBrushData objects. If set, an EmfPlusBlendColors object MUST be specified in the **OptionalData** field of the brush data object.
BrushDataBlendFactorsH 0x00000008	This flag is meaningful in EmfPlusLinearGradientBrushData and EmfPlusPathGradientBrushData objects. If set, an EmfPlusBlendFactors object that specifies a blend pattern along a horizontal gradient MUST be specified in the **OptionalData** field of the brush data object.
BrushDataBlendFactorsV 0x00000010	This flag is meaningful in EmfPlusLinearGradientBrushData objects. If set, an EmfPlusBlendFactors object that specifies a blend pattern along a vertical gradient MUST be specified in the **OptionalData** field of the brush data object.<6>
BrushDataFocusScales 0x00000040	This flag is meaningful in EmfPlusPathGradientBrushData objects. If set, an EmfPlusFocusScaleData object MUST be specified in the **OptionalData** field of the brush data object.
BrushDataIsGammaCorrected 0x00000080	This flag is meaningful in EmfPlusLinearGradientBrushData, EmfPlusPathGradientBrushData, and EmfPlusTextureBrushData objects. If set, the brush MUST already be gamma corrected; that is, output brightness and intensity have been corrected to match the input image.
BrushDataDoNotTransform 0x00000100	This flag is meaningful in EmfPlusTextureBrushData objects. If set, a world space to device space transform SHOULD NOT be applied to the texture brush.

Graphics brushes are specified by EmfPlusBrush objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.2 CustomLineCapData Flags

The CustomLineCapData flags specify data for custom line caps. These flags can be combined to specify multiple options.

Constant/value	Description
CustomLineCapDataFillPath 0x00000001	If set, an EmfPlusFillPath object MUST be specified in the **OptionalData** field of the EmfPlusCustomLineCapData object for filling the custom line cap.
CustomLineCapDataLinePath 0x00000002	If set, an EmfPlusLinePath object MUST be specified in the **OptionalData** field of the EmfPlusCustomLineCapData object for outlining the custom line cap.

Custom graphics line caps are specified by EmfPlusCustomLineCap objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.3 DriverStringOptions Flags

The DriverStringOptions flags specify properties of graphics text positioning and rendering. These flags can be combined to specify multiple options.

Constant/value	Description
DriverStringOptionsCmapLookup 0x00000001	If set, the positions of character glyphs SHOULD be specified in a character map lookup table. If clear, the glyph positions SHOULD be obtained from an array of coordinates.
DriverStringOptionsVertical 0x00000002	If set, the string SHOULD be rendered vertically. If clear, the string SHOULD be rendered horizontally.
DriverStringOptionsRealizedAdvance 0x00000004	If set, character glyph positions SHOULD be calculated relative to the position of the first glyph.<7> If clear, the glyph positions SHOULD be obtained from an array of coordinates.
DriverStringOptionsLimitSubpixel 0x00000008	If set, less memory SHOULD be used to cache anti-aliased glyphs, which produces lower quality text rendering. If clear, more memory SHOULD be used, which produces higher quality text rendering.

Graphics text output is specified in EmfPlusDrawDriverString records.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.4 FontStyle Flags

The FontStyle flags specify styles of graphics font typefaces. These flags can be combined to specify multiple options.

Constant/value	Description
FontStyleBold 0x00000001	If set, the font typeface MUST be rendered with a heavier weight or thickness. If clear, the font typeface MUST be rendered with a normal thickness.
FontStyleItalic	If set, the font typeface MUST be rendered with the vertical stems of the characters at an increased angle or slant relative to the **baseline**.

Constant/value	Description
0x00000002	If clear, the font typeface MUST be rendered with the vertical stems of the characters at a normal angle.
FontStyleUnderline 0x00000004	If set, the font typeface MUST be rendered with a line underneath the baseline of the characters. If clear, the font typeface MUST be rendered without a line underneath the baseline.
FontStyleStrikeout 0x00000008	If set, the font typeface MUST be rendered with a line parallel to the baseline drawn through the middle of the characters. If clear, the font typeface MUST be rendered without a line through the characters.

Graphics font typefaces are specified by EmfPlusFont objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.5 PaletteStyle Flags

The PaletteStyle flags specify properties of graphics palettes. These flags can be combined to specify multiple options.

Constant/value	Description
PaletteStyleHasAlpha 0x00000001	If set, one or more of the palette entries MUST contain alpha transparency information.
PaletteStyleGrayScale 0x00000002	If set, the palette MUST contain only grayscale entries.
PaletteStyleHalftone 0x00000004	If set, the palette MUST contain discrete color values that can be used for **halftoning**.

Graphics palettes are specified by EmfPlusPalette objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.6 PathPointType Flags

The PathPointType flags specify type properties of points on graphics paths. These flags can be combined to specify multiple options.

Constant/value	Description
PathPointTypeDashMode 0x01	Specifies that a line segment that passes through the point is dashed.
PathPointTypePathMarker 0x02	Specifies that the point is a position marker.
PathPointTypeCloseSubpath 0x08	Specifies that the point is the endpoint of a subpath.

Graphics paths are specified by EmfPlusPath objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.7 PenData Flags

The PenData flags specify properties of graphics pens, including the presence of optional data fields. These flags can be combined to specify multiple options.

Constant/value	Description
PenDataTransform 0x00000001	If set, a 2x3 transform matrix MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataStartCap 0x00000002	If set, the style of a starting line cap MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataEndCap 0x00000004	Indicates whether the style of an ending line cap MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataJoin 0x00000008	Indicates whether a line join type MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataMiterLimit 0x00000010	Indicates whether a miter limit MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataLineStyle 0x00000020	Indicates whether a line style MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataDashedLineCap 0x00000040	Indicates whether a dashed line cap MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataDashedLineOffset 0x00000080	Indicates whether a dashed line offset MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataDashedLine 0x00000100	Indicates whether an EmfPlusDashedLineData object MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataNonCenter 0x00000200	Indicates whether a pen alignment MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataCompoundLine 0x00000400	Indicates whether the length and content of a EmfPlusCompoundLineData object are present in the **OptionalData** field of an EmfPlusPenData object.
PenDataCustomStartCap 0x00000800	Indicates whether an EmfPlusCustomStartCapData object MUST be specified in the **OptionalData** field of an EmfPlusPenData object.
PenDataCustomEndCap 0x00001000	Indicates whether an EmfPlusCustomEndCapData object MUST be specified in the **OptionalData** field of an EmfPlusPenData object.

Graphics pens are specified by EmfPlusPen objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.2.8 StringFormat Flags

The StringFormat flags specify options for graphics text layout, including direction, clipping and font handling. These flags can be combined to specify multiple options.

Constant/value	Description
StringFormatDirectionRightToLeft	If set, the reading order of the string SHOULD be right to left. For

Constant/value	Description
0x00000001	horizontal text, this means that characters are read from right to left. For vertical text, this means that columns are read from right to left. If clear, horizontal or vertical text SHOULD be read from left to right.
StringFormatDirectionVertical 0x00000002	If set, individual lines of text SHOULD be drawn vertically on the display device. If clear, individual lines of text SHOULD be drawn horizontally, with each new line below the previous line.
StringFormatNoFitBlackBox 0x00000004	If set, parts of characters MUST be allowed to overhang the text layout rectangle. If clear, characters that overhang the boundaries of the text layout rectangle MUST be repositioned to avoid overhang. An italic, "f" is an example of a character that can have overhanging parts.
StringFormatDisplayFormatControl 0x00000020	If set, control characters SHOULD appear in the output as representative **Unicode** glyphs.
StringFormatNoFontFallback 0x00000400	If set, an alternate font SHOULD be used for characters that are not supported in the requested font. If clear, a character missing from the requested font SHOULD appear as a "font missing" character, which MAY be an open square.
StringFormatMeasureTrailingSpaces 0x00000800	If set, the space at the end of each line MUST be included in measurements of string length. If clear, the space at the end of each line MUST be excluded from measurements of string length.
StringFormatNoWrap 0x00001000	If set, a string that extends past the end of the text layout rectangle MUST NOT be wrapped to the next line. If clear, a string that extends past the end of the text layout rectangle MUST be broken at the last word boundary within the bounding rectangle, and the remainder of the string MUST be wrapped to the next line.
StringFormatLineLimit 0x00002000	If set, whole lines of text SHOULD be output and SHOULD NOT be clipped by the string's layout rectangle. If clear, text layout SHOULD continue until all lines are output, or until additional lines would not be visible as a result of clipping. This flag can be used either to deny or allow a line of text to be partially obscured by a layout rectangle that is not a multiple of line height. For all text to be visible, a layout rectangle at least as tall as the height of one line.
StringFormatNoClip 0x00004000	If set, text extending outside the string layout rectangle SHOULD be allowed to show. If clear, all text that extends outside the layout rectangle SHOULD be clipped.
StringFormatBypassGDI 0x80000000	This flag MAY be used to specify an implementation-specific process for rendering text.<8>

Graphics text layout is specified by EmfPlusStringFormat objects.

See section 2.1.2 for the specification of additional bit flags.

2.1.3 Standard Identifier Constant Types

The Standard Identifier Constants specify GUIDs for EMF+ objects and records.

The following types of standard identifiers are defined:

Name	Section	Description
ImageEffects Identifiers	2.1.3.1	Specifies GUIDs that identify image effects capabilities supported by graphics devices.

2.1.3.1 ImageEffects Identifiers

The ImageEffects identifiers define standard GUIDs for specifying graphics image effects. These identifiers are used by **device drivers** to publish their levels of support for these effects. The identifier constants are defined using the GUID curly-braced string representation ([MS-DTYP] section 2.3.4.3).

Constant/value	Description
BlurEffectGuid {633C80A4-1843-482B-9EF2-BE2834C5FDD4}	Specifies the blur effect.
BrightnessContrastEffectGuid {D3A1DBE1-8EC4-4C17-9F4C-EA97AD1C343D}	Specifies the **brightness contrast effect**.
ColorBalanceEffectGuid {537E597D-251E-48DA-9664-29CA496B70F8}	Specifies the **color balance effect**.
ColorCurveEffectGuid {DD6A0022-58E4-4A67-9D9B-D48EB881A53D}	Specifies the color curve effect.<9>
ColorLookupTableEffectGuid {A7CE72A9-0F7F-40D7-B3CC-D0C02D5C3212}	Specifies the **color lookup table effect**.
ColorMatrixEffectGuid {718F2615-7933-40E3-A511-5F68FE14DD74}	Specifies the **color matrix effect**.
HueSaturationLightnessEffectGuid {8B2DD6C3-EB07-4D87-A5F0-7108E26A9C5F}	Specifies the **hue saturation lightness effect**.
LevelsEffectGuid {99C354EC-2A31-4F3A-8C34-17A803B33A25}	Specifies the **levels effect**.
RedEyeCorrectionEffectGuid {74D29D05-69A4-4266-9549-3CC52836B632}	Specifies the **red-eye correction effect**.
SharpenEffectGuid {63CBF3EE-C526-402C-8F71-62C540BF5142}	Specifies the **sharpen effect**.
TintEffectGuid {1077AF00-2848-4441-9489-44AD4C2D7A2C}	Specifies the **tint effect**.

Image effects identifiers and Image Effects Parameter Blocks are specified by EmfPlusSerializableObject records for EmfPlusDrawImagePoints records.

2.2 EMF+ Objects

This section specifies EMF+ objects, which are grouped into the following categories:

Name	Section	Description
Graphics object types	2.2.1	Specify parameters for graphics output. They are part of the playback device context and are persistent during the playback of an EMF+ metafile.
Structure object types	2.2.2	Specify containers for data structures that are embedded in EMF+ objects and records. Structure objects, unlike graphics objects, are not explicitly created; they are components that make up more complex structures.
Image Effects object types	2.2.3	Specify parameters for graphics image effects, which can be applied to bitmap images.

2.2.1 Graphics Object Types

The Graphics Objects specify parameters for graphics output. They are part of the playback device context and are persistent during the playback of an EMF+ metafile.

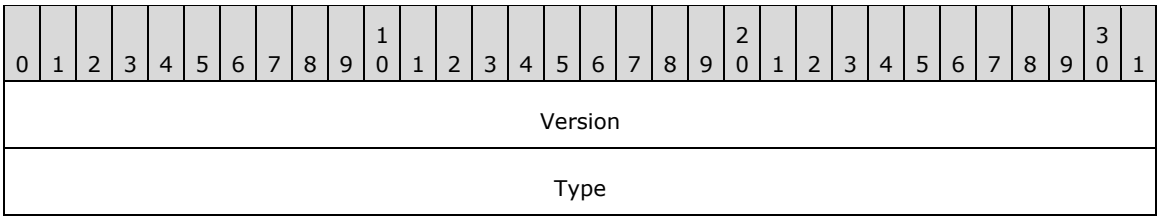
The following types of graphics objects are defined:

Name	Section	Description
EmfPlusBrush	2.2.1.1	Specifies a graphics brush for the filling of figures.
EmfPlusCustomLineCap	2.2.1.2	Specifies the shape to use at the ends of a line drawn by a graphics pen.
EmfPlusFont	2.2.1.3	Specifies properties that determine the appearance of text, including typeface, size, and style.
EmfPlusImage	2.2.1.4	Specifies a graphics image in the form of a bitmap or metafile.
EmfPlusImageAttributes	2.2.1.5	Specifies how bitmap and metafile image colors are manipulated during rendering.
EmfPlusPath	2.2.1.6	Specifies a series of line and curve segments.
EmfPlusPen	2.2.1.7	Specifies a graphics pen for the drawing of lines.
EmfPlusRegion	2.2.1.8	Specifies line and curve segments that define a nonrectilinear shape.
EmfPlusStringFormat	2.2.1.9	Specifies text layout, display manipulations, and language identification.

Graphics objects are explicitly created by EmfPlusObject records, and they can be used in any number of graphics operations. An implementation is responsible for keeping track of graphics objects during playback of the metafile. A conceptual model for managing EMF+ graphics objects is described in Managing Graphics Objects.

2.2.1.1 EmfPlusBrush Object

The EmfPlusBrush object specifies a graphics brush for filling regions.



BrushData (variable)
...

Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

Type (4 bytes): A 32-bit unsigned integer that specifies the type of brush, which determines the contents of the **BrushData** field. This value MUST be defined in the BrushType enumeration.

BrushData (variable): Variable-length data that defines the brush object specified in the **Type** field. The content and format of the data can be different for every brush type.

This object is generic and is used to specify different types of brush data, including the following objects:

- EmfPlusHatchBrushData object
- EmfPlusLinearGradientBrushData object
- EmfPlusPathGradientBrushData object
- EmfPlusSolidBrushData object
- EmfPlusTextureBrushData object

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.2 EmfPlusCustomLineCap Object

The EmfPlusCustomLineCap object specifies the shape to use at the ends of a line drawn by a graphics pen.

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
Version																															
Type																															
CustomLineCapData (variable)																															
...																															

Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

Type (4 bytes): A 32-bit signed integer that specifies the type of custom line cap object, which determines the contents of the **CustomLineCapData** field. This value MUST be defined in the CustomLineCapDataType enumeration.

CustomLineCapData (variable): Variable-length data that defines the custom line cap data object specified in the **Type** field. The content and format of the data can be different for every custom line cap type.

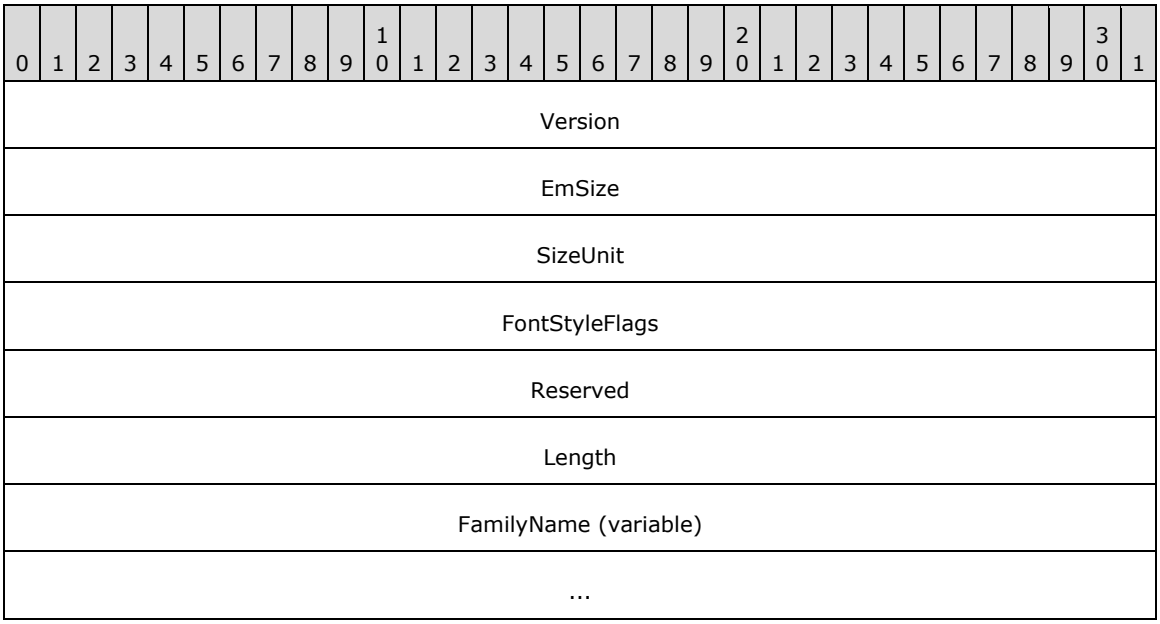
This object is generic and is used to specify different types of custom line cap data, including:

- An EmfPlusCustomLineCapArrowData object; and
- An EmfPlusCustomLineCapData object.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.3 EmfPlusFont Object

The EmfPlusFont object specifies properties that determine the appearance of text, including typeface, size, and style.



Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

EmSize (4 bytes): A 32-bit floating-point value that specifies the em size of the font in units specified by the **SizeUnit** field.

SizeUnit (4 bytes): A 32-bit unsigned integer that specifies the units used for the **EmSize** field. These are typically the units that were employed when designing the font. The value MUST be in the UnitType enumeration.<10>

FontStyleFlags (4 bytes): A 32-bit signed integer that specifies attributes of the character glyphs that affect the appearance of the font, such as bold and italic. This value MUST be composed of FontStyle flags.

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

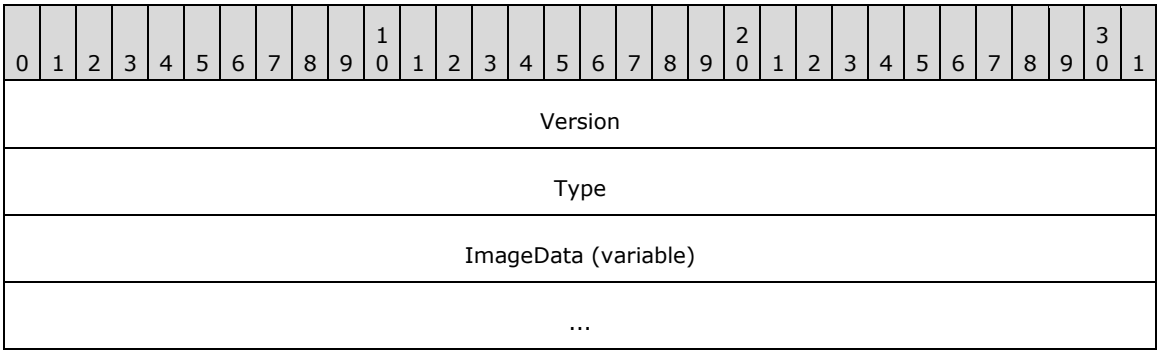
Length (4 bytes): A 32-bit unsigned integer that specifies the number of characters in the **FamilyName** field.

FamilyName (variable): A string of **Length** Unicode characters that contains the name of the font family.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.4 EmfPlusImage Object

The EmfPlusImage object specifies a graphics image in the form of a bitmap or metafile.



Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

Type (4 bytes): A 32-bit unsigned integer that specifies the type of data in the **ImageData** field. This value MUST be defined in the ImageDataType enumeration.

ImageData (variable): Variable-length data that defines the image data specified in the **Type** field. The content and format of the data can be different for every image type.

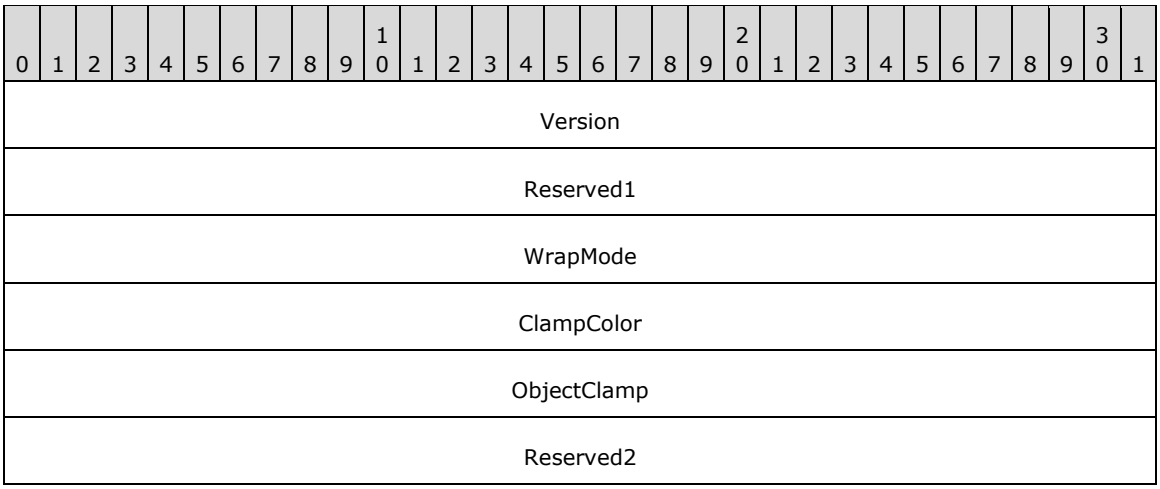
This object is generic and is used to specify different types of image data, including:

- An EmfPlusBitmap object; and
- An EmfPlusMetafile object.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.5 EmfPlusImageAttributes Object

The EmfPlusImageAttributes object specifies how bitmap image colors are manipulated during rendering.



Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

Reserved1 (4 bytes): A 32-bit field that is not used and MUST be ignored.

WrapMode (4 bytes): A 32-bit unsigned integer that specifies how to handle edge conditions with a value from the WrapMode enumeration.

ClampColor (4 bytes): An EmfPlusARGB object that specifies the edge color to use when the **WrapMode** value is **WrapModeClamp**. This color is visible when the source rectangle processed by an EmfPlusDrawImage record is larger than the image itself.

ObjectClamp (4 bytes): A 32-bit signed integer that specifies the object clamping behavior. It is not used until this object is applied to an image being drawn. This value MUST be one of the values defined in the following table.

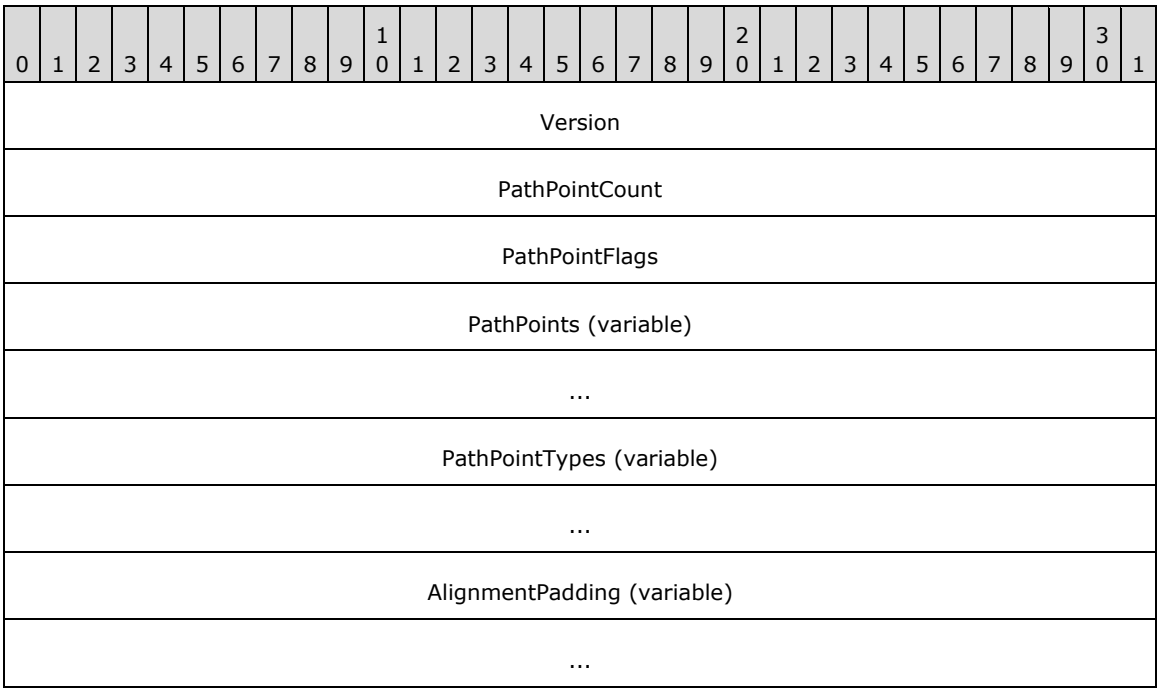
Value	Meaning
RectClamp 0x00000000	The object is clamped to a rectangle.
BitmapClamp 0x00000001	The object is clamped to a bitmap.

Reserved2 (4 bytes): A value that SHOULD be set to zero and MUST be ignored upon receipt.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.6 EmfPlusPath Object

The EmfPlusPath object specifies a series of line and curve segments that form a graphics path. The order for Bezier data points is the start point, control point 1, control point 2, and end point. For more information see [MSDN-DrawBeziers].



Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

PathPointCount (4 bytes): A 32-bit unsigned integer that specifies the number of points and associated point types that are defined by this object.

PathPointFlags (4 bytes): A 32-bit unsigned integer that specifies how to interpret the points and associated point types that are defined by this object.

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
0	0	0	0	0	0	0	0	0	0	0	R	0	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R (1 bit): If this flag is set, the **C** flag is undefined and MUST be ignored. The **R** flag specifies whether the **PathPoints** are relative or absolute locations in the coordinate space, and whether the **PathPointTypes** are run-length encoded. See **PathPoints** and **PathPointTypes** for details.

C (1 bit): If the **R** flag is clear, this flag specifies the type of objects in the **PathPoints** array. See **PathPoints** and **PathPointTypes** for details.

PathPoints (variable): An array of **PathPointCount** points that specify the path. The type of objects in this array is specified by the **PathPointFlags** field, as follows:

- If the **R** flag is set, the points are relative locations specified by EmfPlusPointR objects.
- If the **R** flag is clear and the **C** flag is set, the points are absolute locations specified by EmfPlusPoint objects.
- If the **R** flag is clear and the **C** flag is clear, the points are absolute locations specified by EmfPlusPointF objects.

PathPointTypes (variable): An array of **PathPointCount** objects that specifies how the points in the **PathPoints** field are used to draw the path. The type of objects in this array is specified by the **PathPointFlags** field, as follows:

- If the **R** flag is set, the point types are specified by EmfPlusPathPointTypeRLE objects, which use **run-length encoding (RLE) compression** ([MS-WMF] section 3.1.6).
- If the **R** flag is clear, the point types are specified by EmfPlusPathPointType objects.

AlignmentPadding (variable): An optional array of up to 3 bytes that pads the record so that its total size is a multiple of 4 bytes. This field MUST be ignored.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.7 EmfPlusPen Object

The EmfPlusPen object specifies a graphics pen for the drawing of 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
Version																															
Type																															
PenData (variable)																															

...
BrushObject (variable)
...

Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

Type (4 bytes): This field MUST be set to zero.

PenData (variable): An EmfPlusPenData object that specifies properties of the graphics pen.

BrushObject (variable): An EmfPlusBrush object that specifies a graphics brush associated with the pen.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.8 EmfPlusRegion Object

The EmfPlusRegion object specifies line and curve segments that define a nonrectilinear shape.

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
Version																															
RegionNodeCount																															
RegionNode (variable)																															
...																															

Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

RegionNodeCount (4 bytes): A 32-bit unsigned integer that specifies the number of child nodes in the **RegionNode** field.

RegionNode (variable): An array of **RegionNodeCount**+1 EmfPlusRegionNode objects. Regions are specified as a binary tree of region nodes, and each node MUST either be a terminal node or specify one or two child nodes. RegionNode MUST contain at least one element.

See section 2.2.1 for the specification of additional graphics objects.

2.2.1.9 EmfPlusStringFormat Object

The EmfPlusStringFormat object specifies text layout, display manipulations, and language identification.

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
Version																															

StringFormatFlags
Language
StringAlignment
LineAlign
DigitSubstitution
DigitLanguage
FirstTabOffset
HotkeyPrefix
LeadingMargin
TrailingMargin
Tracking
Trimming
TabStopCount
RangeCount
StringFormatData (variable)
...

Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this object.

StringFormatFlags (4 bytes): A 32-bit unsigned integer that specifies text layout options for formatting, clipping and font handling. This value MUST be composed of StringFormat flags.

Language (4 bytes): An EmfPlusLanguageIdentifier object that specifies the language to use for the string.

StringAlignment (4 bytes): A 32-bit unsigned integer that specifies how to align the string horizontally in the layout rectangle. This value MUST be defined in the StringAlignment enumeration.

LineAlign (4 bytes): A 32-bit unsigned integer that specifies how to align the string vertically in the layout rectangle. This value MUST be defined in the StringAlignment enumeration.

DigitSubstitution (4 bytes): A 32-bit unsigned integer that specifies how to substitute numeric digits in the string according to a locale or language. This value MUST be defined in the StringDigitSubstitution enumeration.

DigitLanguage (4 bytes): An EmfPlusLanguageIdentifier object that specifies the language to use for numeric digits in the string. For example, if this string contains Arabic digits, this field MUST contain a language identifier that specifies an Arabic language.

FirstTabOffset (4 bytes): A 32-bit floating-point value that specifies the number of spaces between the beginning of a text line and the first tab stop.

HotkeyPrefix (4 bytes): A 32-bit signed integer that specifies the type of processing that is performed on a string when a keyboard shortcut prefix (that is, an ampersand) is encountered. Basically, this field specifies whether to display keyboard shortcut prefixes that relate to text. The value MUST be defined in the HotkeyPrefix enumeration.

LeadingMargin (4 bytes): A 32-bit floating-point value that specifies the length of the space to add to the starting position of a string. The default is 1/6 inch; for typographic fonts, the default value is 0.

TrailingMargin (4 bytes): A 32-bit floating-point value that specifies the length of the space to leave following a string. The default is 1/6 inch; for typographic fonts, the default value is 0.

Tracking (4 bytes): A 32-bit floating-point value that specifies the ratio of the horizontal space allotted to each character in a specified string to the font-defined width of the character. Large values for this property specify ample space between characters; values less than 1 can produce character overlap. The default is 1.03; for typographic fonts, the default value is 1.00.

Trimming (4 bytes): Specifies how to trim characters from a string that is too large to fit into a layout rectangle. This value MUST be defined in the StringTrimming enumeration.

TabStopCount (4 bytes): A 32-bit signed integer that specifies the number of tab stops defined in the **StringFormatData** field.

RangeCount (4 bytes): A 32-bit signed integer that specifies the number of EmfPlusCharacterRange object defined in the **StringFormatData** field.

StringFormatData (variable): An EmfPlusStringFormatData object that specifies optional text layout data.

See section 2.2.1 for the specification of additional graphics objects.

2.2.2 Structure Object Types

The Structure Objects specify containers for data structures that are embedded in EMF+ metafile records. Structure objects, unlike graphics objects, are not explicitly created; they are components that make up more complex structures.

The following types of structure objects are defined.

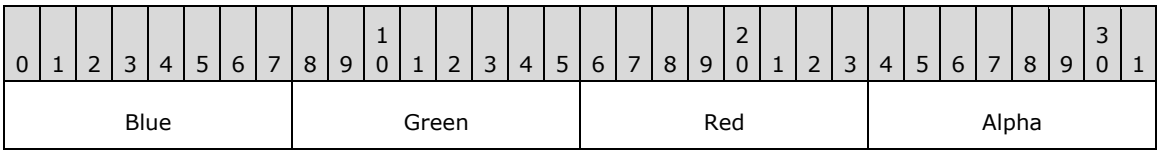
Name	Section	Description
EmfPlusARGB	2.2.2.1	Specifies a color as a combination of **RGB** and alpha.
EmfPlusBitmap	2.2.2.2	Specifies a bitmap image.
EmfPlusBitmapData	2.2.2.3	Specifies a bitmap image with pixel data.
EmfPlusBlendColors	2.2.2.4	Specifies positions and colors for the blend pattern of a gradient brush.
EmfPlusBlendFactors	2.2.2.5	Specifies positions and factors for the blend pattern of a gradient brush.

Name	Section	Description
EmfPlusBoundaryPathData	2.2.2.6	Specifies a path boundary for a gradient brush.
EmfPlusBoundaryPointData	2.2.2.7	Specifies a closed cardinal spline boundary for a gradient brush.
EmfPlusCharacterRange	2.2.2.8	Specifies a range of character positions for a text string.
EmfPlusCompressedImage	2.2.2.10	Specifies an image with compressed data.
EmfPlusCompoundLineData	2.2.2.9	Specifies line and space data for a compound line.
EmfPlusCustomEndCapData	2.2.2.11	Specifies a custom line cap for the end of a line.
EmfPlusCustomLineCapArrowData	2.2.2.12	Specifies adjustable arrow data for a custom line cap.
EmfPlusCustomLineCapData	2.2.2.13	Specifies default data for a custom line cap.
EmfPlusCustomLineCapOptionalData	2.2.2.14	Specifies optional fill and outline data for a custom line cap.
EmfPlusCustomStartCapData	2.2.2.15	Specifies a custom line cap for the start of a line.
EmfPlusDashedLineData	2.2.2.16	Specifies properties of a dashed line for a graphics pen.
EmfPlusFillPath	2.2.2.17	Specifies a graphics path for filling a custom line cap.
EmfPlusFocusScaleData	2.2.2.18	Specifies focus scales for the blend pattern of a path gradient brush.
EmfPlusGraphicsVersion	2.2.2.19	Specifies the version of operating system graphics that is used to create an EMF+ metafile.
EmfPlusHatchBrushData	2.2.2.20	Specifies a hatch pattern for a graphics brush.
EmfPlusInteger7	2.2.2.21	Specifies a 7-bit signed integer in an 8-bit field.
EmfPlusInteger15	2.2.2.22	Specifies a 15-bit signed integer in a 16-bit field.
EmfPlusLanguageIdentifier	2.2.2.23	Specifies language identifiers that correspond to natural languages in a locale.
EmfPlusLinearGradientBrushData	2.2.2.24	Specifies a linear gradient for a graphics brush.
EmfPlusLinearGradientBrushOptionalData	2.2.2.25	Specifies optional data for a linear gradient brush.
EmfPlusLinePath	2.2.2.26	Specifies a graphics path for outlining a custom line cap.
EmfPlusMetafile	2.2.2.27	Specifies a metafile that contains a graphics image.
EmfPlusPalette	2.2.2.28	Specifies the colors that make up a palette.
EmfPlusPathGradientBrushData	2.2.2.29	Specifies a path gradient for a graphics brush.
EmfPlusPathGradientBrushOptionalData	2.2.2.30	Specifies optional data for a path gradient brush.
EmfPlusPathPointType	2.2.2.31	Specifies a type value associated with a point on a graphics path.
EmfPlusPathPointTypeRLE	2.2.2.32	Specifies type values associated with points on a graphics path using RLE compression ([MS-WMF] section 3.1.6).
EmfPlusPenData	2.2.2.33	Specifies properties of a graphics pen.

Name	Section	Description
EmfPlusPenOptionalData	2.2.2.34	Specifies optional data for a graphics pen.
EmfPlusPoint	2.2.2.35	Specifies an ordered pair of integer (X,Y) values that defines an absolute location in a coordinate space.
EmfPlusPointF	2.2.2.36	Specifies an ordered pair of floating-point (X,Y) values that defines an absolute location in a coordinate space.
EmfPlusPointR	2.2.2.37	Specifies an ordered pair of integer (X,Y) values that defines a relative location in a coordinate space.
EmfPlusRect	2.2.2.38	Specifies a rectangle origin, height, and width as 16-bit signed integers.
EmfPlusRectF	2.2.2.39	Specifies a rectangle origin, height, and width as 32-bit floating-point values.
EmfPlusRegionNode	2.2.2.40	Specifies a node of a graphics region.
EmfPlusRegionNodeChildNodes	2.2.2.41	Specifies the child nodes of a graphics region.
EmfPlusRegionNodePath	2.2.2.42	Specifies a graphics path for drawing the boundary of a region node.
EmfPlusSolidBrushData	2.2.2.43	Specifies a solid color for a graphics brush.
EmfPlusStringFormatData	2.2.2.44	Specifies tab stops and character positions for a graphics string.
EmfPlusTextureBrushData	2.2.2.45	Specifies a texture image for a graphics brush.
EmfPlusTextureBrushOptionalData	2.2.2.46	Specifies optional data for a texture brush.
EmfPlusTransformMatrix	2.2.2.47	Specifies a world space to device space transform.

2.2.2.1 EmfPlusARGB Object

The EmfPlusARGB object specifies a color as a combination of red, green, blue. and alpha.



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

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

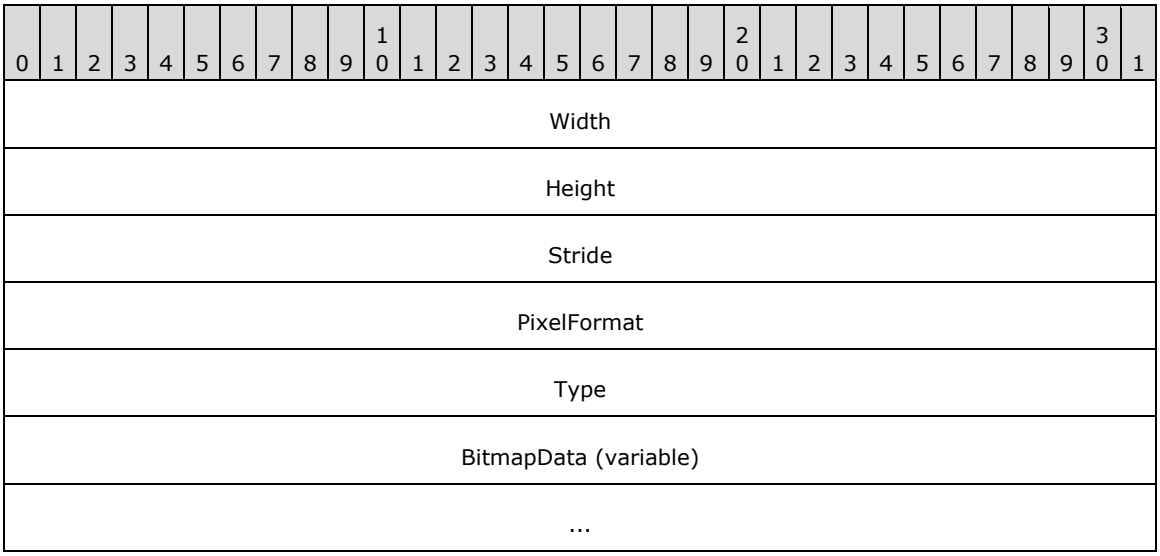
Red (1 byte): An 8-bit unsigned integer that specifies the relative intensity of red.

Alpha (1 byte): An 8-bit unsigned integer that specifies the transparency of the background, ranging from 0 for completely transparent to 0xFF for completely opaque.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.2 EmfPlusBitmap Object

The EmfPlusBitmap object specifies a bitmap that contains a graphics image.



Width (4 bytes): A 32-bit signed integer that specifies the width in pixels of the area occupied by the bitmap.

If the image is compressed, according to the **Type** field, this value is undefined and MUST be ignored.

Height (4 bytes): A 32-bit signed integer that specifies the height in pixels of the area occupied by the bitmap.

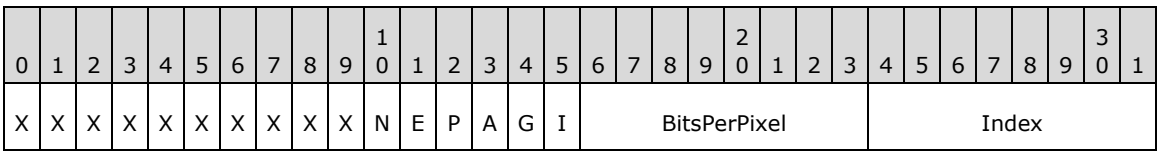
If the image is compressed, according to the **Type** field, this value is undefined and MUST be ignored.

Stride (4 bytes): A 32-bit signed integer that specifies the byte offset between the beginning of one scan-line and the next. This value is the number of bytes per pixel, which is specified in the **PixelFormat** field, multiplied by the width in pixels, which is specified in the **Width** field. The value of this field MUST be a multiple of four.

If the image is compressed, according to the **Type** field, this value is undefined and MUST be ignored.

PixelFormat (4 bytes): A 32-bit unsigned integer that specifies the format of the pixels that make up the bitmap image. The supported pixel formats are specified in the PixelFormat enumeration.

If the image is compressed, according to the **Type** field, this value is undefined and MUST be ignored.



X (1 bit): Reserved and MUST be ignored.

N (1 bit): If set, the pixel format is "canonical", which means that 32 bits per pixel are supported, with 24-bits for color components and an 8-bit alpha channel.

If clear, the pixel format is not canonical.

E (1 bit): If set, the pixel format supports extended colors in 16-bits per channel.

If clear, extended colors are not supported.

P (1 bit): If set, each color component in the pixel has been premultiplied by the pixel's alpha transparency value.

If clear, each color component is multiplied by the pixel's alpha transparency value when the source pixel is blended with the destination pixel.

A (1 bit): If set, the pixel format includes an alpha transparency component.

If clear, the pixel format does not include a component that specifies transparency.

G (1 bit): If set, the pixel format is supported in Windows GDI.

If clear, the pixel format is not supported in Windows GDI.

I (1 bit): If set, the pixel values are indexes into a palette.

If clear, the pixel values are actual colors.

BitsPerPixel (1 byte): The total number of bits per pixel.

Index (1 byte): The pixel format enumeration index.

Type (4 bytes): A 32-bit unsigned integer that specifies the type of data in the **BitmapData** field. This value MUST be defined in the BitmapDataType enumeration.

BitmapData (variable): Variable-length data that defines the bitmap data object specified in the **Type** field. The content and format of the data can be different for every bitmap type.

Graphics images are specified by EmfPlusImage objects. An EmfPlusBitmap object MUST be present in the **ImageData** field of an EmfPlusImage object if ImageTypeBitmap is specified in its **Type** field.

This object is generic and is used to specify different types of bitmap data, including:

- An EmfPlusBitmapData object.
- An EmfPlusCompressedImage object; and

See section 2.2.2 for the specification of additional structure objects.

2.2.2.3 EmfPlusBitmapData Object

The EmfPlusBitmapData object specifies a bitmap image with 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
Colors (variable)																															
...																															
PixelData (variable)																															
...																															

Colors (variable): An optional EmfPlusPalette object, which specifies the palette of colors used in the pixel data. This field MUST be present if the **I** flag is set in the **PixelFormat** field of the EmfPlusBitmap object.

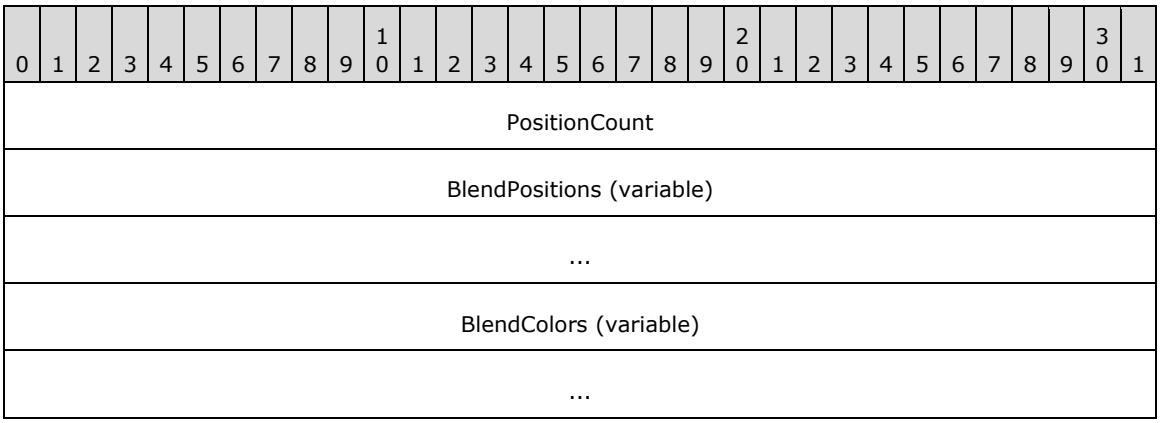
PixelData (variable): An array of bytes that specify the pixel data. The size and format of this data can be computed from fields in the EmfPlusBitmap object, including the pixel format from the PixelFormat enumeration.

Bitmaps are specified by EmfPlusBitmap objects. An EmfPlusBitmapData object MUST be present in the **BitmapData** field of an EmfPlusBitmap object if BitmapDataTypePixel is specified in its **Type** field.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.4 EmfPlusBlendColors Object

The EmfPlusBlendColors object specifies positions and colors for the blend pattern of a gradient brush.



PositionCount (4 bytes): A 32-bit unsigned integer that specifies the number of positions in the **BlendPositions** field and colors in the **BlendColors** field.

BlendPositions (variable): An array of **PositionCount** 32-bit floating-point values that specify proportions of distance along the **gradient line**.

Each element MUST be a number between 0.0 and 1.0 inclusive. For a linear gradient brush, 0.0 represents the starting point and 1.0 represents the ending point. For a path gradient brush, 0.0 represents the midpoint and 1.0 represents an endpoint.

BlendColors (variable): An array of **PositionCount** EmfPlusARGB objects that specify colors at the positions defined in the **BlendPositions** field.

Gradient brushes are specified by EmfPlusLinearGradientBrushData objects and EmfPlusPathGradientBrushData objects. Blend patterns are used to smoothly shade the interiors of shapes filled by gradient brushes. and can be defined by arrays of positions and colors or positions and factors. Positions and factors are specified by EmfPlusBlendFactors objects.

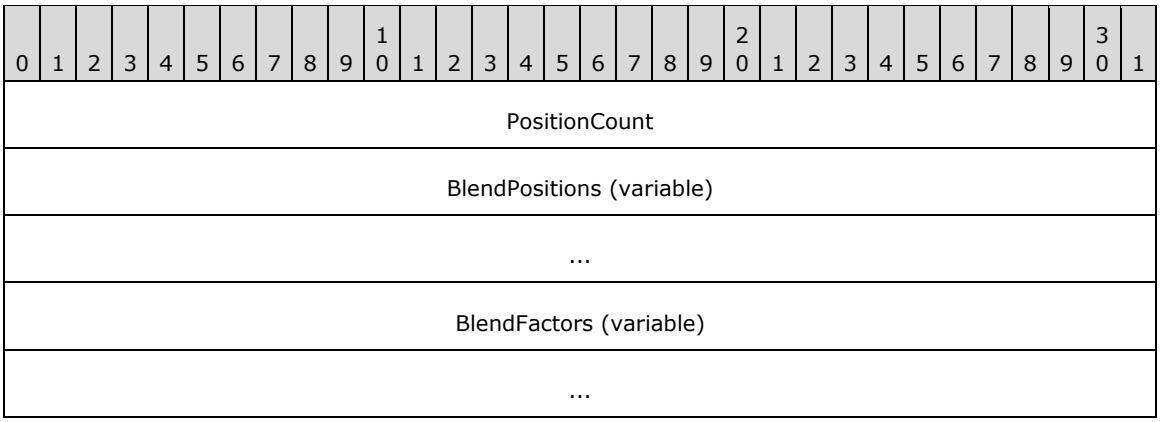
An EmfPlusBlendColors object MUST be present in the **OptionalData** field of an EmfPlusLinearGradientBrushData object, if the **BrushDataPresetColors** flag is set in its **BrushDataFlags** field.

An EmfPlusBlendColors object MUST be present in the **OptionalData** field of an EmfPlusPathGradientBrushData object, if the **BrushDataPresetColors** flag is set in its **BrushDataFlags** field.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.5 EmfPlusBlendFactors Object

The EmfPlusBlendFactors object specifies positions and factors for the blend pattern of a gradient brush.



PositionCount (4 bytes): A 32-bit unsigned integer that specifies the number of positions in the **BlendPositions** field and factors in the **BlendFactors** field.

BlendPositions (variable): An array of **PositionCount** 32-bit floating-point values that specify proportions of distance along the gradient line.

Each value MUST be a number between 0.0 and 1.0 inclusive. There MUST be at least two positions specified: the first position, which is always 0.0f, and the last position, which is always 1.0f. Each position in **BlendPositions** is generally greater than the preceding position. For a linear gradient brush, 0.0 represents the starting point and 1.0 represents the ending point. For a path gradient brush, 0.0 represents the midpoint and 1.0 represents an endpoint.

BlendFactors (variable): An array of **PositionCount** 32-bit floating point values that specify proportions of colors at the positions defined in the **BlendPositions** field. Each value MUST be a number between 0.0 and 1.0 inclusive.

For a linear gradient brush, 0.0 represents 0% starting color and 100% ending color, and 1.0 represents 100% starting color and 0% ending color. For a path gradient brush, 0.0 represents 0% midpoint color and 100% endpoint color, and 1.0 represents 100% midpoint color and 0% endpoint color.

For example, if a linear gradient brush specifies a position of 0.2 and a factor of 0.3 along a gradient line that is 100 pixels long, the color that is 20 pixels along that line consists of 30 percent starting color and 70 percent ending color.

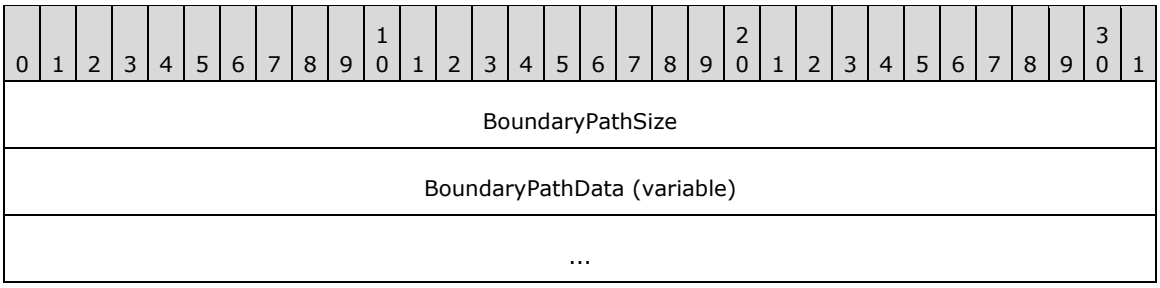
Gradient brushes are specified by EmfPlusLinearGradientBrushData objects and EmfPlusPathGradientBrushData objects. Blend patterns are used to smoothly shade the interiors of shapes filled by gradient brushes. and can be defined by arrays of positions and colors or positions and factors. Positions and colors are specified by EmfPlusBlendColors objects.

An EmfPlusBlendFactors object MUST be present in the **OptionalData** field of an EmfPlusLinearGradientBrushData or EmfPlusPathGradientBrushData object if either of the flags **BrushDataBlendFactorsH** or **BrushDataBlendFactorsV** is set in its **BrushDataFlags** field.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.6 EmfPlusBoundaryPathData Object

The EmfPlusBoundaryPathData object specifies a graphics path boundary for a gradient brush.



BoundaryPathSize (4 bytes): A 32-bit signed integer that specifies the size in bytes of the **BoundaryPathData** field.

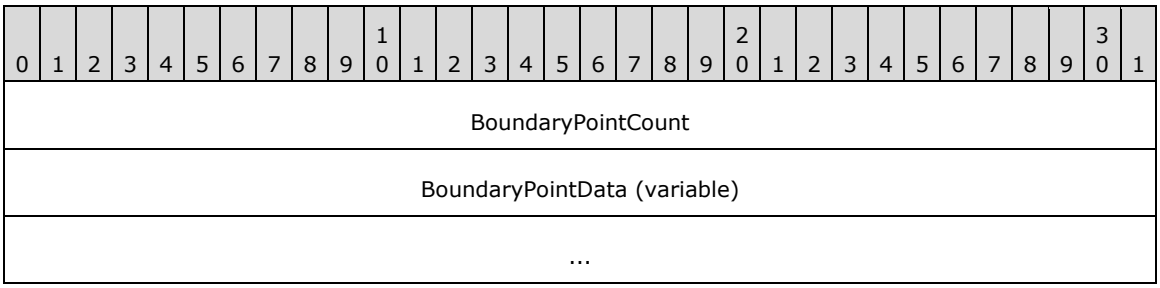
BoundaryPathData (variable): An EmfPlusPath object, which specifies the boundary of the brush.

Boundary path data is specified in the **BoundaryData** field of an EmfPlusPathGradientBrushData object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.7 EmfPlusBoundaryPointData Object

The EmfPlusBoundaryPointData object specifies a closed cardinal spline boundary for a gradient brush.



BoundaryPointCount (4 bytes): A 32-bit signed integer that specifies the number of points in the **BoundaryPointData** field.

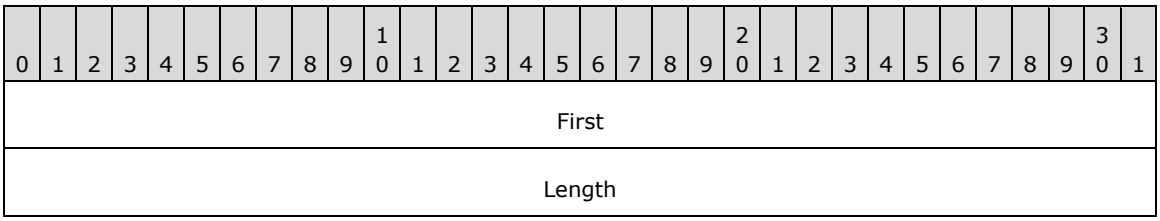
BoundaryPointData (variable): An array of **BoundaryPointCount** EmfPlusPointF objects that specify the boundary of the brush.

Boundary point data is specified in the **BoundaryData** field of an EmfPlusPathGradientBrushData object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.8 EmfPlusCharacterRange Object

The EmfPlusCharacterRange object specifies a range of character positions for a text string.



First (4 bytes): A 32-bit signed integer that specifies the first position of this range.

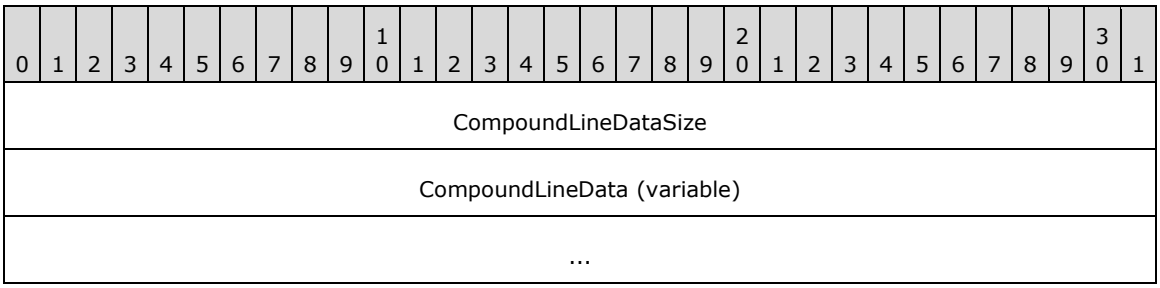
Length (4 bytes): A 32-bit signed integer that specifies the number of positions in this range.

Graphics strings are specified by EmfPlusStringFormat objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.9 EmfPlusCompoundLineData Object

The EmfPlusCompoundLineData object specifies line and space data for a compound line.



CompoundLineDataSize (4 bytes): A 32-bit unsigned integer that specifies the number of elements in the **CompoundLineData** field.

CompoundLineData (variable): An array of **CompoundLineDataSize** floating-point values that specify the compound line of a pen. The elements MUST be in increasing order, and their values MUST be between 0.0 and 1.0, inclusive.

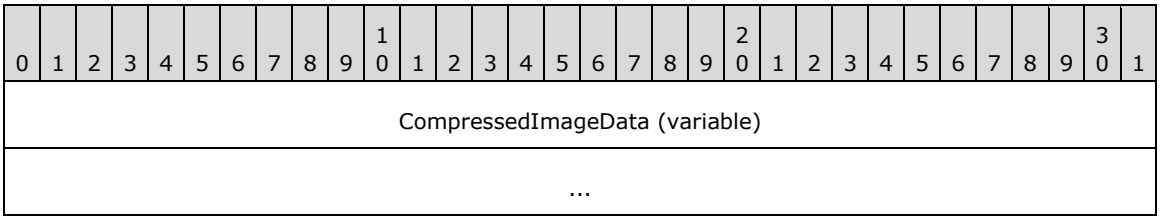
Graphics pens are specified by EmfPlusPen objects. An EmfPlusCompoundLineData object MUST be present in the **OptionalData** field of an EmfPlusPenData object, if the **PenDataCompoundLineData** flag is set in its **PenDataFlags** field.

A compound line is made up of a pattern of alternating parallel lines and spaces of varying widths. The values in the array specify the starting points of each component of the compound line relative to the total width. The first value specifies where the first line component begins as a fraction of the distance across the width of the pen. The second value specifies where the first space component begins as a fraction of the distance across the width of the pen. The final value in the array specifies where the last line component ends.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.10 EmfPlusCompressedImage Object

The EmfPlusCompressedImage object specifies an image with compressed data.



CompressedImageData (variable): An array of bytes, which specify the compressed image. The type of compression MUST be determined from the data itself.

Bitmaps are specified by EmfPlusBitmap objects. An EmfPlusCompressedImage object MUST be present in the **BitmapData** field of an EmfPlusBitmap object if BitmapDataTypeCompressed is specified in its **Type** field.

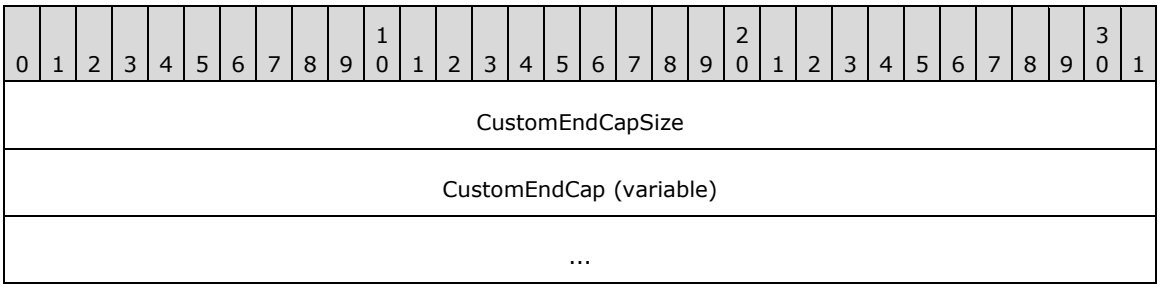
This object is generic and is used for different types of compressed data, including:

- **Exchangeable Image File Format (EXIF)** [EXIF];
- **Graphics Interchange Format (GIF)** [GIF];
- **Joint Photographic Experts Group (JPEG)** [JFIF];
- **Portable Network Graphics (PNG)** [RFC2083] [W3C-PNG]; and
- **Tag Image File Format (TIFF)** [RFC3302] [TIFF].

See section 2.2.2 for the specification of additional structure objects.

2.2.2.11 EmfPlusCustomEndCapData Object

The EmfPlusCustomEndCapData object specifies a custom line cap for the end of a line.



CustomEndCapSize (4 bytes): A 32-bit unsigned integer that specifies the size in bytes of the **CustomEndCap** field.

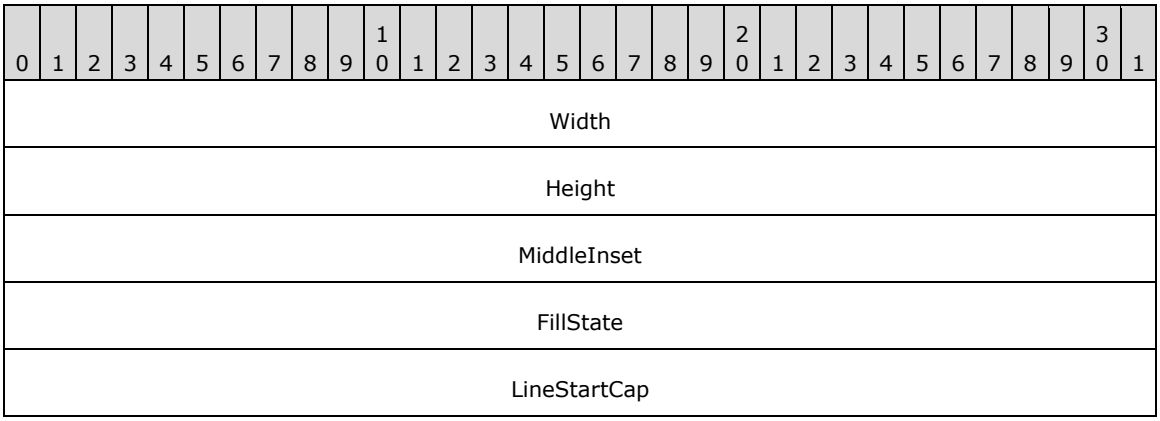
CustomEndCap (variable): A custom line cap that defines the shape to draw at the end of a line. It can be any of various shapes, including a square, circle, or diamond.

Custom line caps are specified by EmfPlusCustomLineCap objects. An EmfPlusCustomEndCapData object MUST be present in the **OptionalData** field of an EmfPlusPenData object, if the **PenDataEndCap** flag is set in its **PenDataFlags** field.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.12 EmfPlusCustomLineCapArrowData Object

The EmfPlusCustomLineCapArrowData object specifies adjustable arrow data for a custom line cap.



LineEndCap
LineJoin
LineMiterLimit
WidthScale
FillHotSpot
...
LineHotSpot
...

Width (4 bytes): A 32-bit floating-point value that specifies the width of the arrow cap.

The width of the arrow cap is scaled by the width of the EmfPlusPen object that is used to draw the line being capped. For example, when drawing a capped line with a pen that has a width of 5 pixels, and the adjustable arrow cap object has a width of 3, the actual arrow cap is drawn 15 pixels wide.

Height (4 bytes): A 32-bit floating-point value that specifies the height of the arrow cap.

The height of the arrow cap is scaled by the width of the EmfPlusPen object that is used to draw the line being capped. For example, when drawing a capped line with a pen that has a width of 5 pixels, and the adjustable arrow cap object has a height of 3, the actual arrow cap is drawn 15 pixels high.

MiddleInset (4 bytes): A 32-bit floating-point value that specifies the number of pixels between the outline of the arrow cap and the fill of the arrow cap.

FillState (4 bytes): A 32-bit Boolean value that specifies whether the arrow cap is filled. If the arrow cap is not filled, only the outline is drawn.

LineStartCap (4 bytes): A 32-bit unsigned integer that specifies the value in the LineCap enumeration that indicates the line cap to be used at the start of the line to be drawn.

LineEndCap (4 bytes): A 32-bit unsigned integer that specifies the value in the LineCap enumeration that indicates the line cap to be used at the end of the line to be drawn.

LineJoin (4 bytes): A 32-bit unsigned integer that specifies the value in the LineJoin enumeration that specifies how to join two lines that are drawn by the same pen and whose ends meet. At the intersection of the two line ends, a line join makes the connection look more continuous.

LineMiterLimit (4 bytes): A 32-bit floating-point value that specifies the limit of the thickness of the join on a mitered corner by setting the maximum allowed ratio of **miter length** to line width.

WidthScale (4 bytes): A 32-bit floating-point value that specifies the amount by which to scale an EmfPlusCustomLineCap object with respect to the width of the graphics pen that is used to draw the lines.

FillHotSpot (8 bytes): An EmfPlusPointF object that is not currently used. It MUST be set to {0.0, 0.0}.

LineHotSpot (8 bytes): An EmfPlusPointF object that is not currently used. It MUST be set to {0.0, 0.0}.

Custom line caps are specified by EmfPlusCustomLineCap objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.13 EmfPlusCustomLineCapData Object

The EmfPlusCustomLineCapData object specifies default data for a custom line cap.

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
CustomLineCapDataFlags																															
BaseCap																															
BaseInset																															
StrokeStartCap																															
StrokeEndCap																															
StrokeJoin																															
StrokeMiterLimit																															
WidthScale																															
FillHotSpot																															
...																															
StrokeHotSpot																															
...																															
OptionalData (variable)																															
...																															

CustomLineCapDataFlags (4 bytes): A 32-bit unsigned integer that specifies the data in the **OptionalData** field. This value MUST be composed of CustomLineCapData flags.

BaseCap (4 bytes): A 32-bit unsigned integer that specifies the value from the LineCap enumeration on which the custom line cap is based.

BaseInset (4 bytes): A 32-bit floating-point value that specifies the distance between the beginning of the line cap and the end of the line.

StrokeStartCap (4 bytes): A 32-bit unsigned integer that specifies the value in the LineCap enumeration that indicates the line cap used at the start of the line to be drawn.

StrokeEndCap (4 bytes): A 32-bit unsigned integer that specifies the value in the LineCap enumeration that indicates what line cap is to be used at the end of the line to be drawn.

StrokeJoin (4 bytes): A 32-bit unsigned integer that specifies the value in the LineJoin enumeration, which specifies how to join two lines that are drawn by the same pen and whose ends meet. At the intersection of the two line ends, a line join makes the connection look more continuous.

StrokeMiterLimit (4 bytes): A 32-bit floating-point value that contains the limit of the thickness of the join on a mitered corner by setting the maximum allowed ratio of miter length to line width.

WidthScale (4 bytes): A 32-bit floating-point value that specifies the amount by which to scale the custom line cap with respect to the width of the EmfPlusPen object that is used to draw the lines.

FillHotSpot (8 bytes): An EmfPlusPointF object that is not currently used. It MUST be set to {0.0, 0.0}.

StrokeHotSpot (8 bytes): An EmfPlusPointF object that is not currently used. It MUST be set to {0.0, 0.0}.

OptionalData (variable): An optional EmfPlusCustomLineCapOptionalData object that specifies additional data for the custom graphics line cap. The specific contents of this field are determined by the value of the **CustomLineCapDataFlags** field.

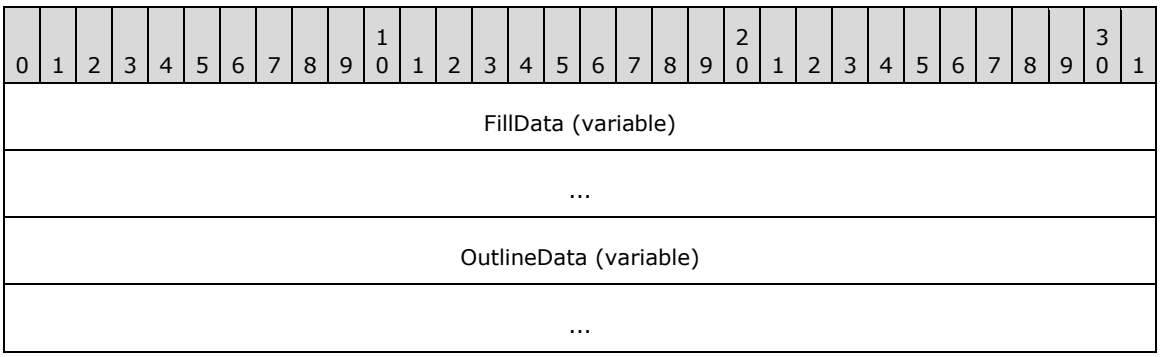
Custom line caps are specified by EmfPlusCustomLineCap objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.14 EmfPlusCustomLineCapOptionalData Object

The EmfPlusCustomLineCapOptionalData object specifies optional fill and outline data for a custom line cap.

Note: Each field specified for this object is optional and might not be present in the **OptionalData** field of an EmfPlusCustomLineCapData object, depending on the CustomLineCapData flags set in its **CustomLineCapDataFlags** field. Although it is not practical to represent every possible combination of fields present or absent, this section specifies their relative order in the object. The implementer is responsible for determining which fields are actually present in a given metafile record, and for unmarshaling the data for individual fields separately and appropriately.



FillData (variable): An optional EmfPlusFillPath object that specifies the path for filling a custom graphics line cap. This field MUST be present if the **CustomLineCapDataFillPath** flag is set in the **CustomLineCapDataFlags** field of the EmfPlusCustomLineCapData object.

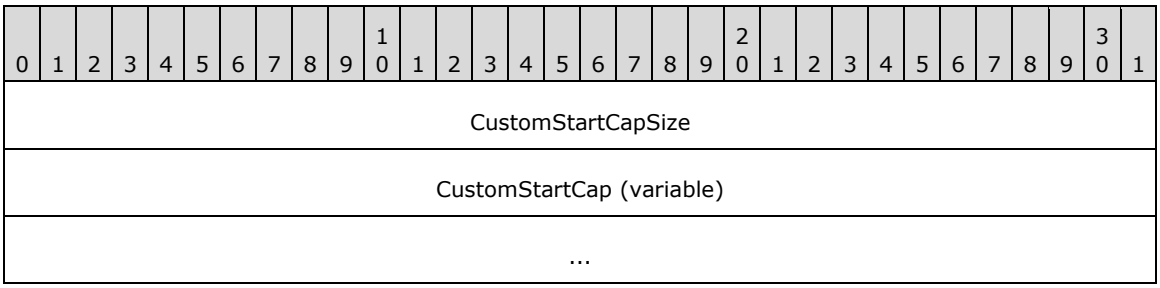
OutlineData (variable): An optional EmfPlusLinePath object that specifies the path for outlining a custom graphics line cap. This field MUST be present if the **CustomLineCapDataLinePath** flag is set in the **CustomLineCapDataFlags** field of the EmfPlusCustomLineCapData object.

Custom line caps are specified by EmfPlusCustomLineCap objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.15 EmfPlusCustomStartCapData Object

The EmfPlusCustomStartCapData object specifies a custom line cap for the start of a line.



CustomStartCapSize (4 bytes): A 32-bit unsigned integer that specifies the size in bytes of the **CustomStartCap** field.

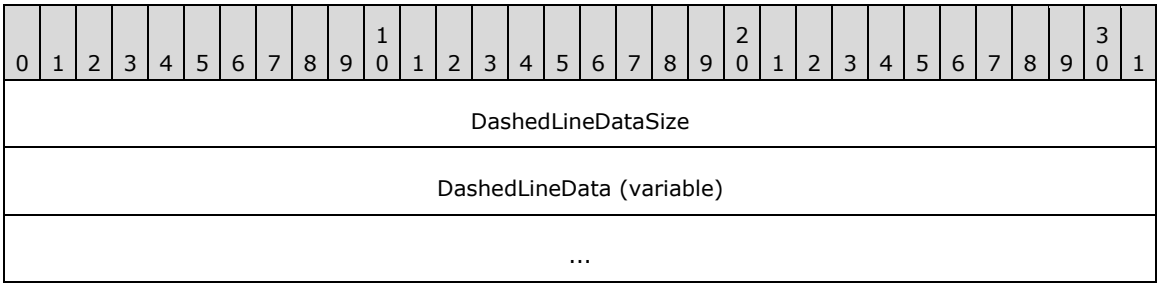
CustomStartCap (variable): A custom graphics line cap that defines the shape to draw at the start of a line. It can be any of various shapes, including a square, circle or diamond.

Custom line caps are specified by EmfPlusCustomLineCap objects. If the **PenDataStartCap** flag is set in its **PenDataFlags** field, an EmfPlusCustomStartCapData object MUST be present in the **OptionalData** field of an EmfPlusPenData object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.16 EmfPlusDashedLineData Object

The EmfPlusDashedLineData object specifies properties of a dashed line for a graphics pen.



DashedLineDataSize (4 bytes): A 32-bit unsigned integer that specifies the number of elements in the **DashedLineData** field.

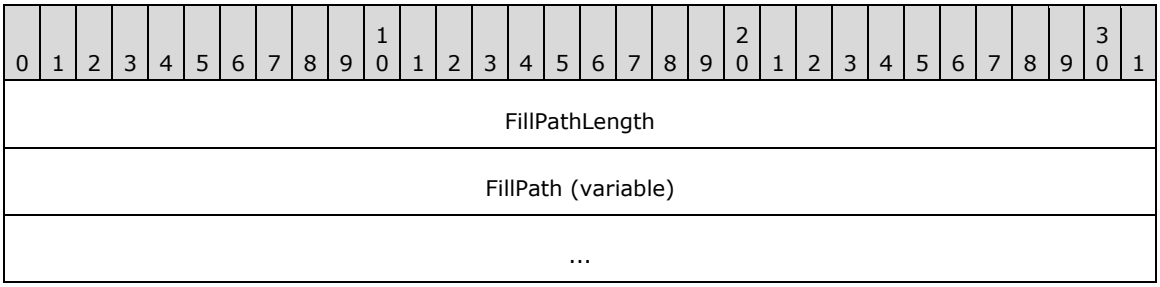
DashedLineData (variable): An array of **DashedLineDataSize** floating-point values that specify the lengths of the dashes and spaces in a dashed line.

Graphics pens are specified by EmfPlusPen objects. An EmfPlusDashedLineData object MUST be present in the **OptionalData** field of an EmfPlusPenData object, if the **PenDataDashedLine** flag is set in its **PenDataFlags** field.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.17 EmfPlusFillPath Object

The EmfPlusFillPath object specifies a graphics path for filling a custom line cap.



FillPathLength (4 bytes): A 32-bit signed integer that specifies the length in bytes of the **FillPath** field.

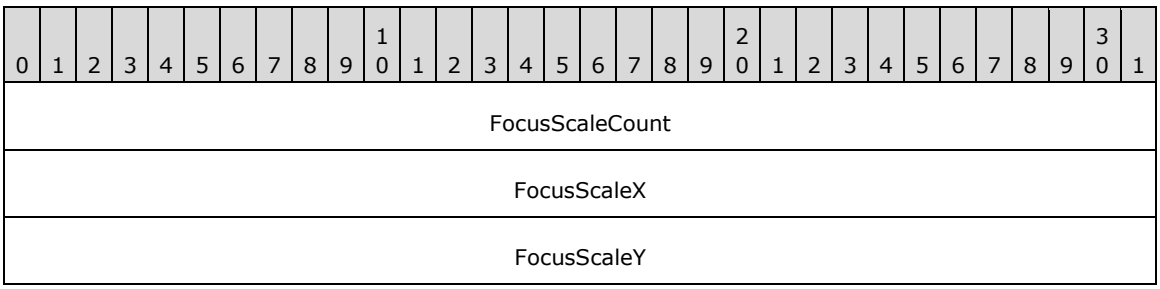
FillPath (variable): An EmfPlusPath, which specifies the area to fill.

Custom line caps are specified by EmfPlusCustomLineCap objects. An EmfPlusFillPath object MUST be present if the **CustomLineCapDataFillPath** flag is set in the **CustomLineCapDataFlags** field of an EmfPlusCustomLineCapData object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.18 EmfPlusFocusScaleData Object

The EmfPlusFocusScaleData object specifies focus scales for the blend pattern of a path gradient brush.



FocusScaleCount (4 bytes): A 32-bit unsigned integer that specifies the number of focus scales. This value MUST be 2.

FocusScaleX (4 bytes): A floating-point value that defines the horizontal focus scale. The focus scale MUST be a value between 0.0 and 1.0, exclusive.

FocusScaleY (4 bytes): A floating-point value that defines the vertical focus scale. The focus scale MUST be a value between 0.0 and 1.0, exclusive.

By default, the center color of a path gradient brush is displayed only at the center point of an area bounded by a path. Focus scales specify an inner path inside that area, and the center color is displayed everywhere inside it. The inner path is the boundary path scaled by horizontal and vertical scale factors.

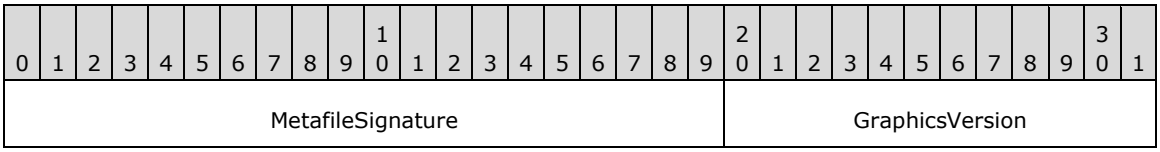
For example, focus scales of {0.2, 0.3} specifies a path that is the boundary path scaled by a factor of 0.2 horizontally and 0.3 vertically. The area inside the scaled path MUST be filled with the center color. Between the inner and outer boundaries, the color MUST change gradually from the center color to the boundary color.

An EmfPlusFocusScaleData object MUST be present in the **OptionalData** field of an EmfPlusPathGradientBrushData object, if the **BrushDataFocusScales** flag is set in its **BrushDataFlags** field.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.19 EmfPlusGraphicsVersion Object

The EmfPlusGraphicsVersion object specifies the version of operating system graphics that is used to create an EMF+ metafile.



MetafileSignature (20 bits): A value that identifies the type of metafile. The value for an EMF+ metafile is 0xDBC01.

GraphicsVersion (12 bits): The version of operating system graphics. This value MUST be defined in the GraphicsVersion enumeration.<11>

Graphics versions are vendor-extensible; however, to ensure inter-operability, any such extension MUST be implemented in both clients and servers of EMF+ metafiles.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.20 EmfPlusHatchBrushData Object

The EmfPlusHatchBrushData object specifies a hatch pattern for a graphics brush.

HatchStyle (4 bytes): A 32-bit unsigned integer that specifies the brush hatch style. It MUST be defined in the HatchStyle enumeration.

ForeColor (4 bytes): A 32-bit EmfPlusARGB object that specifies the color used to draw the lines of the hatch pattern.

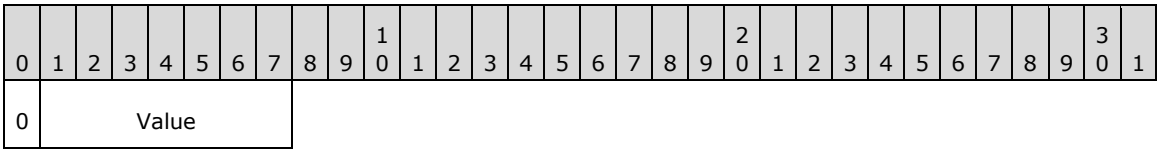
BackColor (4 bytes): A 32-bit EmfPlusARGB object that specifies the color used to paint the background of the hatch pattern.

Graphics brushes are specified by EmfPlusBrush objects. A hatch brush paints a background and draws a pattern of lines, dots, dashes, squares, and crosshatch lines over this background. The hatch brush defines two colors: one for the background and one for the pattern over the background. The color of the background is called the background color, and the color of the pattern is called the foreground color.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.21 EmfPlusInteger7 Object

The EmfPlusInteger7 object specifies a 7-bit signed integer in an 8-bit field.



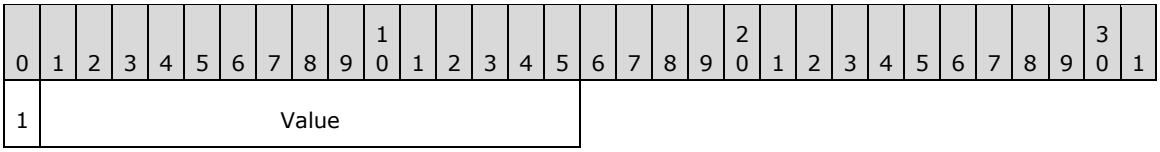
Value (7 bits): A 7-bit signed integer between -64 and 63, inclusive.

EmfPlusInteger7 objects are used to specify point coordinates in EmfPlusPointR object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.22 EmfPlusInteger15 Object

The EmfPlusInteger15 object specifies a 15-bit signed integer in a 16-bit field.



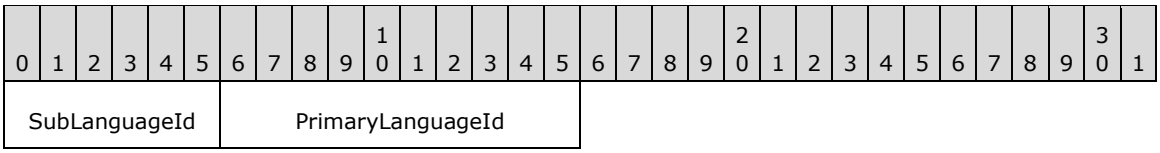
Value (15 bits): A 15-bit signed integer between -16,384 and 16,383, inclusive.

EmfPlusInteger15 objects are used to specify point coordinates in EmfPlusPointR object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.23 EmfPlusLanguageIdentifier Object

The EmfPlusLanguageIdentifier object specifies a language identifier that corresponds to the natural language in a locale, including countries, geographical regions, and administrative districts. Each language identifier is an encoding of a primary language value and sublanguage value.



SubLanguageId (6 bits): The country, geographic region or administrative district for the natural language specified in the **PrimaryLanguageId** field.

Sublanguage identifiers are vendor-extensible. Vendor-defined sublanguage identifiers MUST be in the range 0x20 to 0x3F, inclusive.

PrimaryLanguageId (10 bits): The natural language.

Primary language identifiers are vendor-extensible. Vendor-defined primary language identifiers MUST be in the range 0x0200 to 0x03FF, inclusive.

The 16-bit encoded language identifier value MUST be defined in the LanguageIdentifier enumeration.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.24 EmfPlusLinearGradientBrushData Object

The EmfPlusLinearGradientBrushData object specifies a linear gradient for a graphics brush.

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
BrushDataFlags																															
WrapMode																															
RectF (16 bytes)																															
...																															
...																															
StartColor																															
EndColor																															
Reserved1																															
Reserved2																															
OptionalData (variable)																															
...																															

BrushDataFlags (4 bytes): A 32-bit unsigned integer that specifies the data in the **OptionalData** field. This value MUST be composed of BrushData flags. The following flags are relevant to a linear gradient brush:

Name	Value
BrushDataTransform	0x00000002
BrushDataPresetColors	0x00000004
BrushDataBlendFactorsH	0x00000008
BrushDataBlendFactorsV	0x00000010
BrushDataIsGammaCorrected	0x00000080

WrapMode (4 bytes): A 32-bit signed integer from the WrapMode enumeration that specifies whether to paint the area outside the boundary of the brush. When painting outside the boundary, the wrap mode specifies how the color gradient is repeated.

RectF (16 bytes): An EmfPlusRectF object that specifies the starting and ending points of the gradient line. The upper-left corner of the rectangle is the starting point. The lower-right corner is the ending point.

StartColor (4 bytes): An EmfPlusARGB object, which specifies the color at the starting boundary point of the linear gradient brush.

EndColor (4 bytes): An EmfPlusARGB object that specifies the color at the ending boundary point of the linear gradient brush.

Reserved1 (4 bytes): This field is reserved and SHOULD<12> be ignored.

Reserved2 (4 bytes): This field is reserved and SHOULD<13> be ignored.

OptionalData (variable): An optional EmfPlusLinearGradientBrushOptionalData object that specifies additional data for the linear gradient brush. The specific contents of this field are determined by the value of the **BrushDataFlags** field.

Graphics brushes are specified by EmfPlusBrush objects. A linear gradient brush paints a color gradient in which the color changes gradually along a gradient line from a starting boundary point to an ending boundary point, which are specified by the diagonal of a rectangle in the **RectF** field.

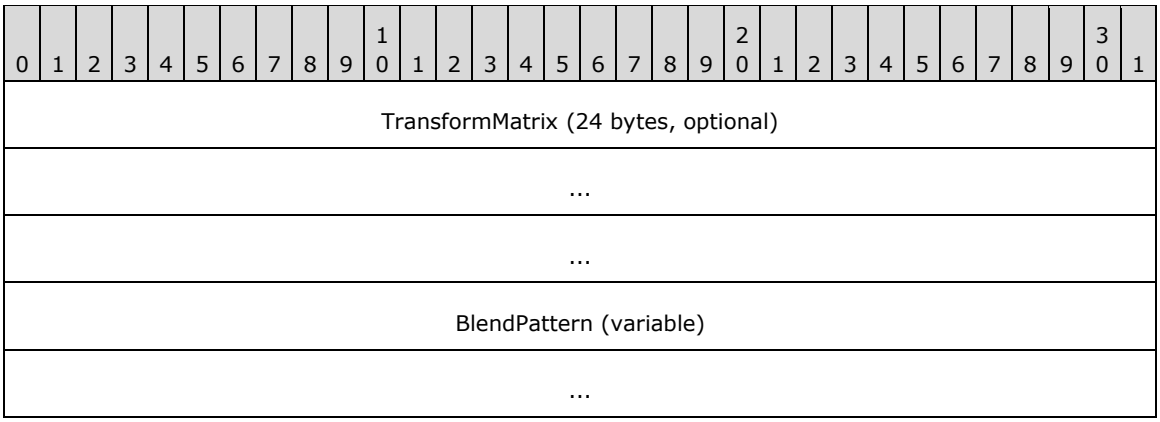
Gamma correction controls the overall brightness and intensity of an image. Uncorrected images can look either bleached out or too dark. Varying the amount of gamma correction changes not only the brightness but also the ratios of red to green to blue. The need for gamma correction arises because an output device might not render colors in the same intensity as the input image.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.25 EmfPlusLinearGradientBrushOptionalData Object

The EmfPlusLinearGradientBrushOptionalData object specifies optional data for a linear gradient brush.

Note: Each field of this object is optional and might not be present in the **OptionalData** field of an EmfPlusLinearGradientBrushData object, depending on the BrushData flags set in its **BrushDataFlags** field. Although it is not practical to represent every possible combination of fields present or absent, this section specifies their relative order in the object. The implementer is responsible for determining which fields are actually present in a given metafile record, and for unmarshaling the data for individual fields separately and appropriately.



TransformMatrix (24 bytes): An optional EmfPlusTransformMatrix object that specifies a world space to device space transform for the linear gradient brush. This field MUST be present if the **BrushDataTransform** flag is set in the **BrushDataFlags** field of the EmfPlusLinearGradientBrushData object.

BlendPattern (variable): An optional blend pattern for the linear gradient brush. If this field is present, it MUST contain either an EmfPlusBlendColors object, or one or two EmfPlusBlendFactors objects, but it MUST NOT contain both. The table below shows the valid combinations of EmfPlusLinearGradientBrushData BrushData flags and the corresponding blend patterns:

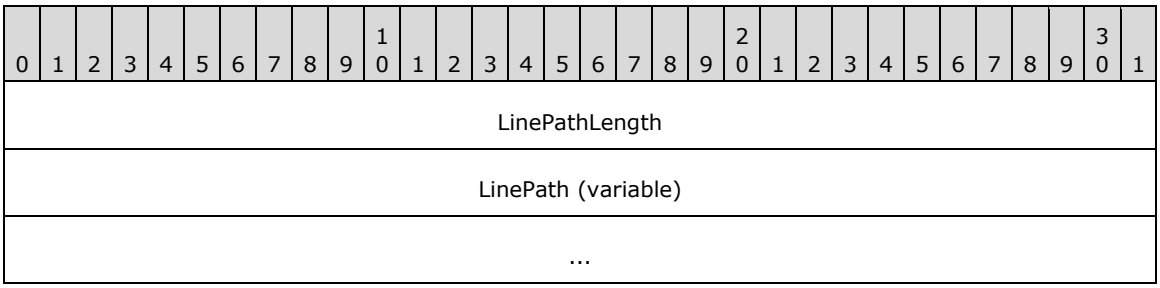
PresetColors	BlendFactorsH	BlendFactorsV	Blend Pattern
Clear	Clear	Clear	This field MUST NOT be present in the EmfPlusLinearGradientBrushOptionalData object.
Set	Clear	Clear	An EmfPlusBlendColors object MUST be present.
Clear	Set	Clear	An EmfPlusBlendFactors object along the horizontal gradient line MUST be present.
Clear	Clear	Set	An EmfPlusBlendFactors object along the vertical gradient line MUST be present.<14>
Clear	Set	Set	An EmfPlusBlendFactors object along the vertical gradient line and an EmfPlusBlendFactors object along the horizontal gradient line MUST be present.<15>

Graphics brushes are specified by EmfPlusBrush objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.26 EmfPlusLinePath Object

The EmfPlusLinePath object specifies a graphics path for outlining a custom line cap.



LinePathLength (4 bytes): A 32-bit signed integer that defines the length in bytes of the **LinePath** field.

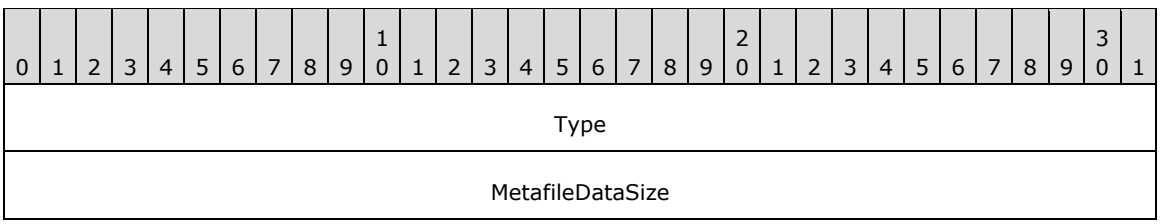
LinePath (variable): An EmfPlusPath object that defines the outline.

Custom line caps are specified by EmfPlusCustomLineCap objects. An EmfPlusLinePath object MUST be present if the **CustomLineCapDataLinePath** flag is set in the **CustomLineCapDataFlags** field of an EmfPlusCustomLineCapData object.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.27 EmfPlusMetafile Object

The EmfPlusMetafileData object specifies a metafile that contains a graphics image.



MetafileData (variable)
...

Type (4 bytes): A 32-bit unsigned integer that specifies the type of metafile that is embedded in the **MetafileData** field. This value MUST be defined in the MetafileDataType enumeration.

MetafileDataSize (4 bytes): A 32-bit unsigned integer that specifies the size in bytes of the metafile data in the **MetafileData** field.

MetafileData (variable): Variable-length data that specifies the embedded metafile. The content and format of the data can be different for each metafile type.

Graphics images are specified by EmfPlusImage objects. An EmfPlusMetafile object MUST be present in the **ImageData** field of an EmfPlusImage object if ImageTypeMetafile is specified in its **Type** field.

This object is generic and is used for different types of data, including:

- A WMF metafile [MS-WMF];
- A WMF metafile which can be placed;
- An EMF metafile [MS-EMF];
- An EMF+ metafile that specifies graphics operations with EMF+ records only; and
- An EMF+ metafile that specifies graphics operations with both EMF+ and EMF records ([MS-EMF] section 2.3).

See section 2.2.2 for the specification of additional structure objects.

2.2.2.28 EmfPlusPalette Object

The EmfPlusPalette object specifies the colors that make up a palette.

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
PaletteStyleFlags																															
PaletteCount																															
PaletteEntries (variable)																															
...																															

PaletteStyleFlags (4 bytes): A 32-bit unsigned integer that specifies the attributes of data in the palette. This value MUST be composed of PaletteStyle flags.

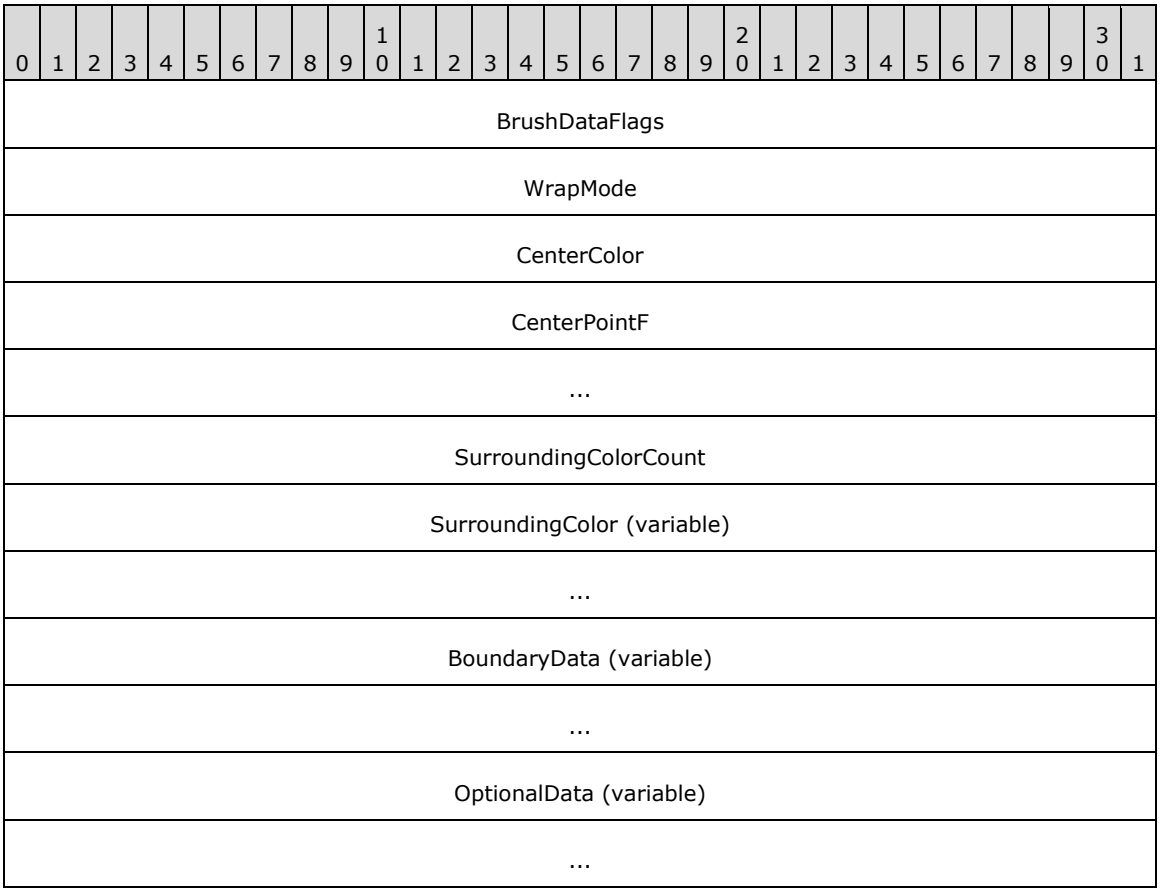
PaletteCount (4 bytes): A 32-bit unsigned integer that specifies the number of entries in the **PaletteEntries** array.

PaletteEntries (variable): An array of **PaletteCount** EmfPlusARGB objects that specify the data in the palette.

See section 2.2.2 for the specification of additional graphics objects.

2.2.2.29 EmfPlusPathGradientBrushData Object

The EmfPlusPathGradientBrushData object specifies a path gradient for a graphics brush.



BrushDataFlags (4 bytes): A 32-bit unsigned integer that specifies the data in the **OptionalData** field. This value MUST be composed of BrushData flags. The following flags are relevant to a path gradient brush:

Name	Value
BrushDataPath	0x00000001
BrushDataTransform	0x00000002
BrushDataPresetColors	0x00000004
BrushDataBlendFactorsH	0x00000008
BrushDataFocusScales	0x00000040
BrushDataIsGammaCorrected	0x00000080

WrapMode (4 bytes): A 32-bit signed integer from the WrapMode enumeration that specifies whether to paint the area outside the boundary of the brush. When painting outside the boundary, the wrap mode specifies how the color gradient is repeated.

CenterColor (4 bytes): An EmfPlusARGB object that specifies the center color of the path gradient brush, which is the color that appears at the center point of the brush. The color of the brush

changes gradually from the boundary color to the center color as it moves from the boundary to the center point.

CenterPointF (8 bytes): An EmfPlusPointF object that specifies the center point of the path gradient brush, which can be any location inside or outside the boundary. The color of the brush changes gradually from the boundary color to the center color as it moves from the boundary to the center point.

SurroundingColorCount (4 bytes): An unsigned 32-bit integer that specifies the number of colors specified in the **SurroundingColor** field. The surrounding colors are colors specified for discrete points on the boundary of the brush.

SurroundingColor (variable): An array of **SurroundingColorCount** EmfPlusARGB objects that specify the colors for discrete points on the boundary of the brush.

BoundaryData (variable): The boundary of the path gradient brush, which is specified by either a path or a closed cardinal spline. If the **BrushDataPath** flag is set in the **BrushDataFlags** field, this field MUST contain an EmfPlusBoundaryPathData object; otherwise, this field MUST contain an EmfPlusBoundaryPointData object.

OptionalData (variable): An optional EmfPlusPathGradientBrushOptionalData object that specifies additional data for the path gradient brush. The specific contents of this field are determined by the value of the **BrushDataFlags** field.

Graphics brushes are specified by EmfPlusBrush objects. A path gradient brush paints a color gradient in which the color changes gradually along a gradient line from the center point outward to the boundary, which are specified by either a closed cardinal spline or a path in the **BoundaryData** field.

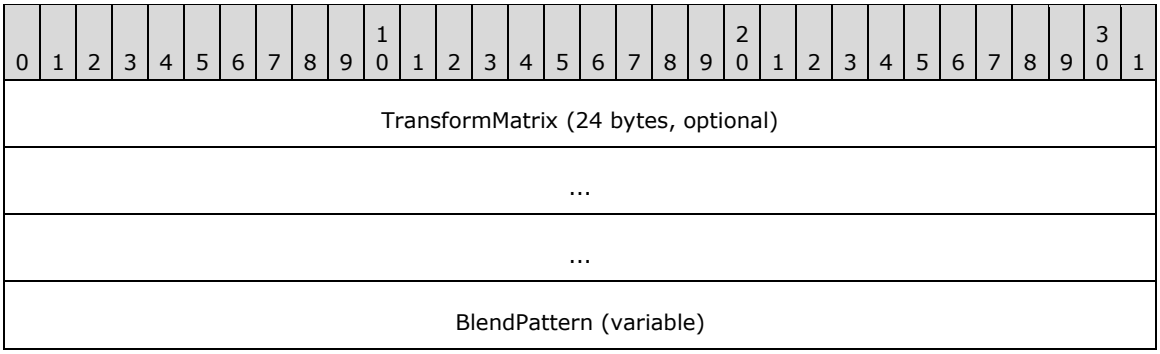
Gamma correction controls the overall brightness and intensity of an image. Uncorrected images can look either bleached out or too dark. Varying the amount of gamma correction changes not only the brightness but also the ratios of red to green to blue. The need for gamma correction arises because an output device might not render colors in the same intensity as the input image.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.30 EmfPlusPathGradientBrushOptionalData Object

The EmfPlusPathGradientBrushOptionalData object specifies optional data for a path gradient brush.

Note: Each field of this object is optional and might not be present in the **OptionalData** field of an EmfPlusPathGradientBrushData object, depending on the BrushData flags set in its **BrushDataFlags** field. Although it is not practical to represent every possible combination of fields present or absent, this section specifies their relative order in the object. The implementer is responsible for determining which fields are actually present in a given metafile record, and for unmarshaling the data for individual fields separately and appropriately.



...
FocusScaleData (optional)
...
...

TransformMatrix (24 bytes): An optional EmfPlusTransformMatrix object that specifies a world space to device space transform for the path gradient brush. This field MUST be present if the **BrushDataTransform** flag is set in the **BrushDataFlags** field of the EmfPlusPathGradientBrushData object.

BlendPattern (variable): An optional blend pattern for the path gradient brush. If this field is present, it MUST contain either an EmfPlusBlendColors object, or an EmfPlusBlendFactors object, but it MUST NOT contain both. The table below shows the valid combinations of EmfPlusPathGradientBrushData BrushData flags and the corresponding blend patterns:

PresetColors	BlendFactorsH	Blend Patterns
Clear	Clear	This field MUST NOT be present.
Set	Clear	An EmfPlusBlendColors object MUST be present.
Clear	Set	An EmfPlusBlendFactors object MUST be present.

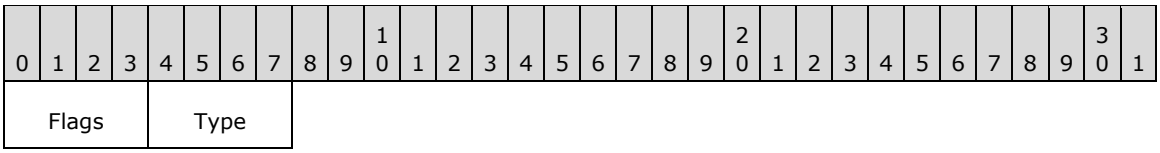
FocusScaleData (12 bytes): An optional EmfPlusFocusScaleData object that specifies focus scales for the path gradient brush. This field MUST be present if the **BrushDataFocusScales** flag is set in the **BrushDataFlags** field of the EmfPlusPathGradientBrushData object.

Graphics brushes are specified by EmfPlusBrush objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.31 EmfPlusPathPointType Object

The EmfPlusPathPointType object specifies a type value associated with a point on a graphics path.



Flags (4 bits): A 4-bit flag field that specifies properties of the path point. This value MUST be one or more of the PathPointType flags.

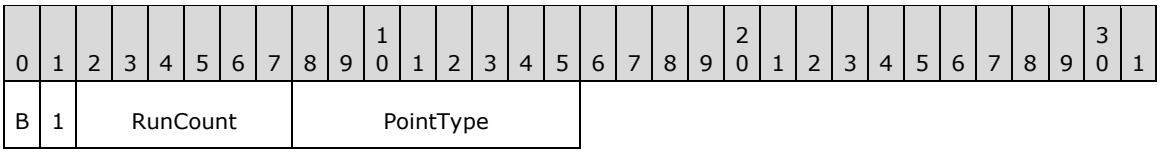
Type (4 bits): A 4-bit unsigned integer path point type. This value MUST be defined in the PathPointType enumeration.

Graphics paths are specified by EmfPlusPath objects. Every point on a graphics path MUST have a type value associated with it.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.32 EmfPlusPathPointTypeRLE Object

The EmfPlusPathPointTypeRLE object specifies type values associated with points on a graphics path using RLE compression ([MS-WMF] section 3.1.6).



B (1 bit): If set, the path points are on a Bezier curve.

If clear, the path points are on a graphics line.

RunCount (6 bits): The run count, which is the number of path points to be associated with the type in the **PointType** field.

PointType (1 byte): An EmfPlusPathPointType object that specifies the type to associate with the path points.

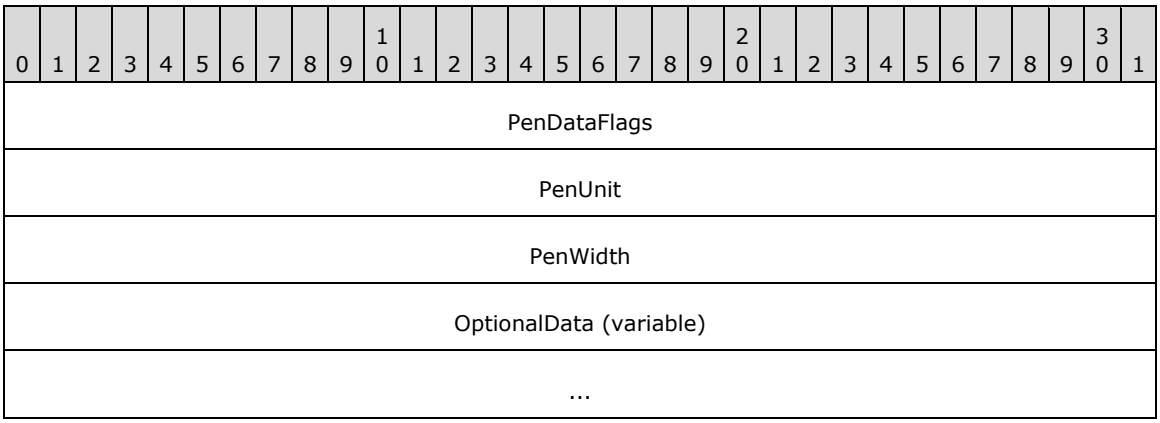
Graphics paths are specified by EmfPlusPath objects. Every point on a graphics path MUST have a type value associated with it.

RLE compression makes it possible to specify an arbitrary number of identical values without a proportional increase in storage requirements.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.33 EmfPlusPenData Object

The EmfPlusPenData object specifies properties of a graphics pen.



PenDataFlags (4 bytes): A 32-bit unsigned integer that specifies the data in the **OptionalData** field. This value MUST be composed of PenData flags.

PenUnit (4 bytes): A 32-bit unsigned integer that specifies the measuring units for the pen. The value MUST be from the UnitType enumeration.

PenWidth (4 bytes): A 32-bit floating-point value that specifies the width of the line drawn by the pen in the units specified by the **PenUnit** field. If a zero width is specified, a minimum value is used, which is determined by the units.

OptionalData (variable): An optional EmfPlusPenOptionalData object that specifies additional data for the pen object. The specific contents of this field are determined by the value of the **PenDataFlags** field.

Graphics pens are specified by EmfPlusPen objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.34 EmfPlusPenOptionalData Object

The EmfPlusPenOptionalData object specifies optional data for a graphics pen.

Note: Each field of this object is optional and might not be present in the **OptionalData** field of an EmfPlusPenData object, depending on the PenData flags set in its **PenDataFlags** field. Although it is not practical to represent every possible combination of fields present or absent, this section specifies their relative order in the object. The implementer is responsible for determining which fields are actually present in a given metafile record, and for unmarshaling the data for individual fields separately and appropriately.

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
TransformMatrix (24 bytes, optional)																															
...																															
...																															
StartCap (optional)																															
EndCap (optional)																															
Join (optional)																															
MiterLimit (optional)																															
LineStyle (optional)																															
DashedLineCapType (optional)																															
DashOffset (optional)																															
DashedLineData (variable)																															
...																															
PenAlignment (optional)																															
CompoundLineData (variable)																															
...																															
CustomStartCapData (variable)																															

...
CustomEndCapData (variable)
...

TransformMatrix (24 bytes): An optional EmfPlusTransformMatrix object that specifies a world space to device space transform for the pen. This field MUST be present if the **PenDataTransform** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

StartCap (4 bytes): An optional 32-bit signed integer that specifies the shape for the start of a line in the **CustomStartCapData** field. This field MUST be present if the **PenDataStartCap** flag is set in the **PenDataFlags** field of the EmfPlusPenData object, and the value MUST be defined in the LineCapType enumeraiton.

EndCap (4 bytes): An optional 32-bit signed integer that specifies the shape for the end of a line in the **CustomEndCapData** field. This field MUST be present if the **PenDataEndCap** flag is set in the **PenDataFlags** field of the EmfPlusPenData object, and the value MUST be defined in the LineCapType enumeration.

Join (4 bytes): An optional 32-bit signed integer that specifies how to join two lines that are drawn by the same pen and whose ends meet. This field MUST be present if the **PenDataJoin** flag is set in the **PenDataFlags** field of the EmfPlusPenData object, and the value MUST be defined in the LineJoinType enumeration.

MiterLimit (4 bytes): An optional 32-bit floating-point value that specifies the miter limit, which is the maximum allowed ratio of miter length to line width. The miter length is the distance from the intersection of the line walls on the inside the join to the intersection of the line walls outside the join. The miter length can be large when the angle between two lines is small. This field MUST be present if the **PenDataMiterLimit** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

LineStyle (4 bytes): An optional 32-bit signed integer that specifies the style used for lines drawn with this pen object. This field MUST be present if the **PenDataLineStyle** flag is set in the **PenDataFlags** field of the EmfPlusPenData object, and the value MUST be defined in the LineStyle enumeration.

DashedLineCapType (4 bytes): An optional 32-bit signed integer that specifies the shape for both ends of each dash in a dashed line. This field MUST be present if the **PenDataDashedLineCap** flag is set in the **PenDataFlags** field of the EmfPlusPenData object, and the value MUST be defined in the DashedLineCapType enumeration.

DashOffset (4 bytes): An optional 32-bit floating-point value that specifies the distance from the start of a line to the start of the first space in a dashed line pattern. This field MUST be present if the **PenDataDashedLineOffset** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

DashedLineData (variable): An optional EmfPlusDashedLineData object that specifies the lengths of dashes and spaces in a custom dashed line. This field MUST be present if the **PenDataDashedLine** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

PenAlignment (4 bytes): An optional 32-bit signed integer that specifies the distribution of the pen width with respect to the coordinates of the line being drawn. This field MUST be present if the **PenDataNonCenter** flag is set in the **PenDataFlags** field of the EmfPlusPenData object, and the value MUST be defined in the PenAlignment enumeration.

For example, consider the placement of a line. If the starting and ending coordinates of the line are defined, it is possible to think of a theoretical line between the two points that is zero width.

Center alignment means that the pen width is distributed as evenly as possible on either side of that theoretical line.

CompoundLineData (variable): An optional EmfPlusCompoundLineData object that specifies an array of floating-point values that define the compound line of a pen, which is made up of parallel lines and spaces. This field MUST be present if the **PenDataCompoundLine** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

CustomStartCapData (variable): An optional EmfPlusCustomStartCapData object that defines the custom start-cap shape, which is the shape to use at the start of a line drawn with this pen. It can be any of various shapes, such as a square, circle, or diamond. This field MUST be present if the **PenDataCustomStartCap** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

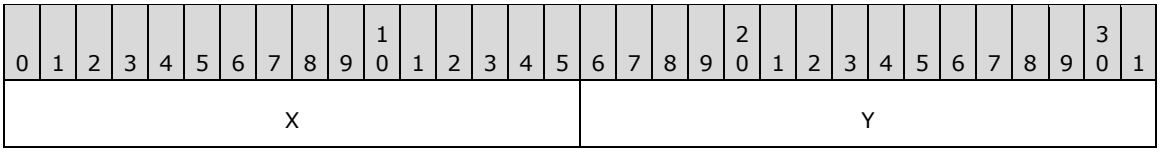
CustomEndCapData (variable): An optional EmfPlusCustomEndCapData object that defines the custom end-cap shape, which is the shape to use at the end of a line drawn with this pen. It can be any of various shapes, such as a square, circle, or diamond. This field MUST be present if the **PenDataCustomEndCap** flag is set in the **PenDataFlags** field of the EmfPlusPenData object.

Graphics pens are specified by EmfPlusPen objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.35 EmfPlusPoint Object

The EmfPlusPoint object specifies an ordered pair of integer (X,Y) values that define an absolute location in a coordinate space.



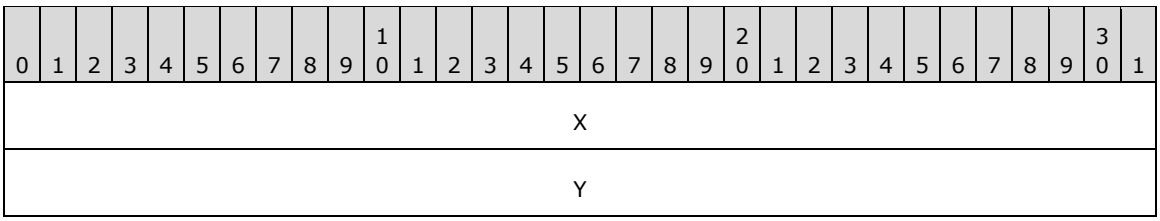
X (2 bytes): A 16-bit signed integer that defines the horizontal coordinate.

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

See section 2.2.2 for the specification of additional structure objects.

2.2.2.36 EmfPlusPointF Object

The EmfPlusPointF object specifies an ordered pair of floating-point (X,Y) values that define an absolute location in a coordinate space.



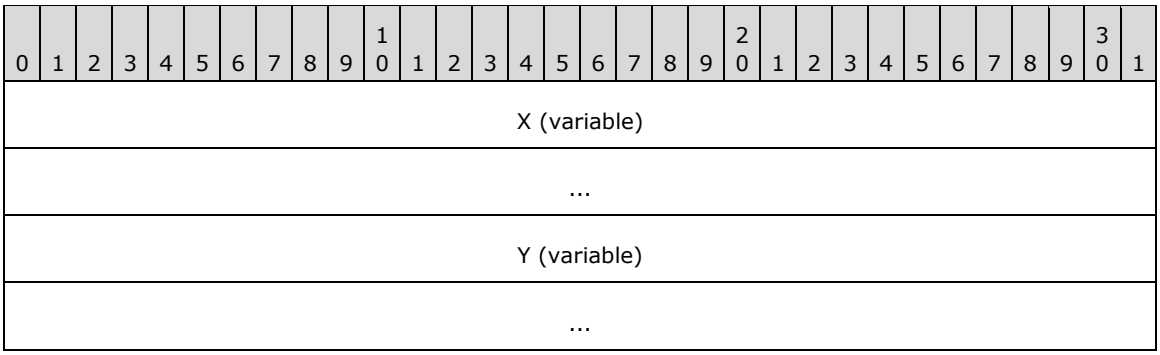
X (4 bytes): A 32-bit floating-point value that specifies the horizontal coordinate.

Y (4 bytes): A 32-bit floating-point value that specifies the vertical coordinate.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.37 EmfPlusPointR Object

The EmfPlusPointR object specifies an ordered pair of integer (X,Y) values that define a relative location in a coordinate space.



X (variable): A signed integer that specifies the horizontal coordinate. This value MUST be specified by either an EmfPlusInteger7 object or an EmfPlusInteger15 object.

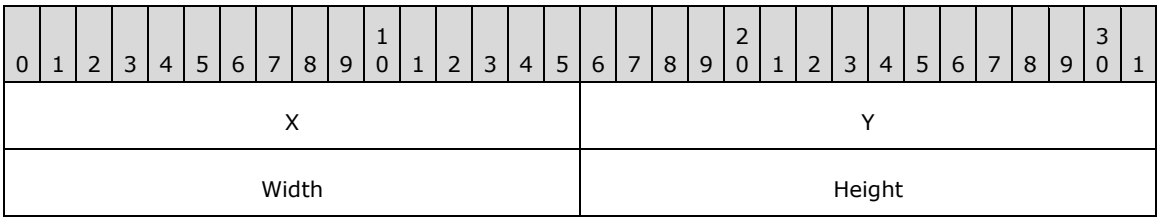
Y (variable): A signed integer that specifies the vertical coordinate. This value MUST be specified by either an EmfPlusInteger7 object or an EmfPlusInteger15 object.

Note: The object that specifies the horizontal coordinate is not required to be the same type as the object that specifies the vertical coordinate; that is, one can be 7 bits and the other can be 15 bits.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.38 EmfPlusRect Object

The EmfPlusRect object specifies a rectangle origin, height, and width as 16-bit signed integers.



X (2 bytes): A 16-bit signed integer that specifies the horizontal coordinate of the upper-left corner of the rectangle.

Y (2 bytes): A 16-bit signed integer that specifies the vertical coordinate of the upper-left corner of the rectangle.

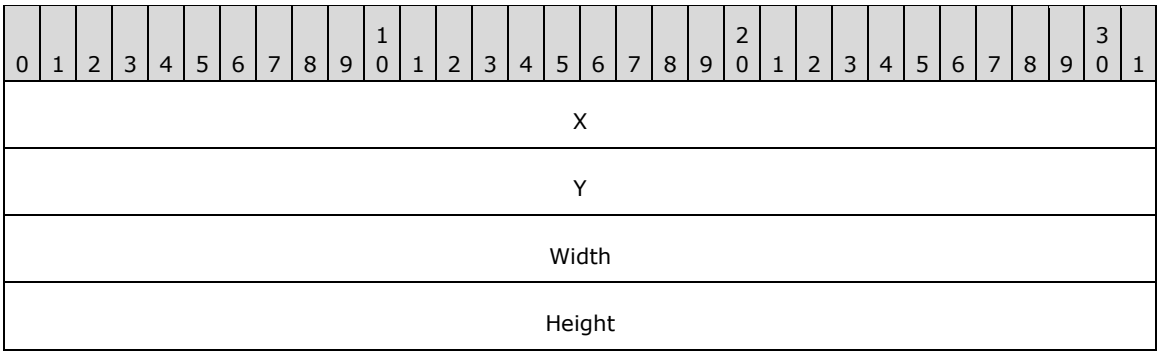
Width (2 bytes): A 16-bit signed integer that specifies the width of the rectangle.

Height (2 bytes): A 16-bit signed integer that specifies the height of the rectangle.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.39 EmfPlusRectF Object

The EmfPlusRectF object specifies a rectangle's origin, height, and width as 32-bit floating-point values.



X (4 bytes): A 32-bit floating-point value that specifies the horizontal coordinate of the upper-left corner of the rectangle.

Y (4 bytes): A 32-bit floating-point value that specifies the vertical coordinate of the upper-left corner of the rectangle.

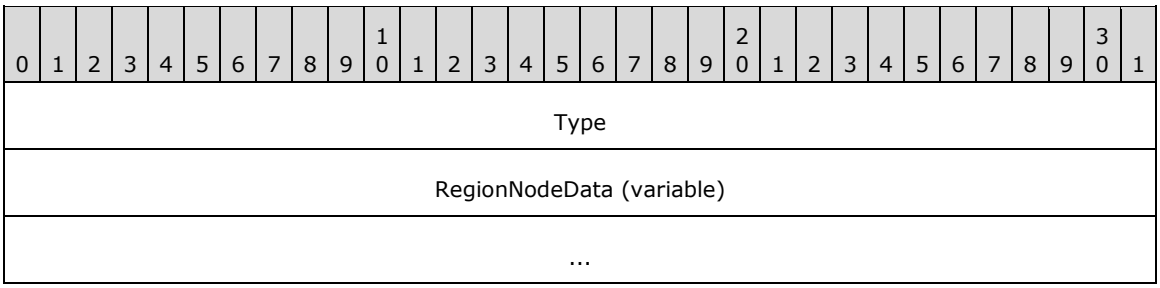
Width (4 bytes): A 32-bit floating-point value that specifies the width of the rectangle.

Height (4 bytes): A 32-bit floating-point value that specifies the height of the rectangle.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.40 EmfPlusRegionNode Object

The EmfPlusRegionNode object specifies nodes of a graphics region.



Type (4 bytes): A 32-bit unsigned integer that specifies the type of data in the **RegionNodeData** field. This value MUST be defined in the RegionNodeDataType enumeration.

RegionNodeData (variable): Optional, variable-length data that defines the region node data object specified in the **Type** field. The content and format of the data can be different for every region node type. This field MUST NOT be present if the node type is RegionNodeDataTypeEmpty or RegionNodeDataTypeInfinite.

Graphics regions are specified by EmfPlusRegion objects, which define a binary tree of region nodes. Each node MUST either be a terminal node or specify additional region nodes.

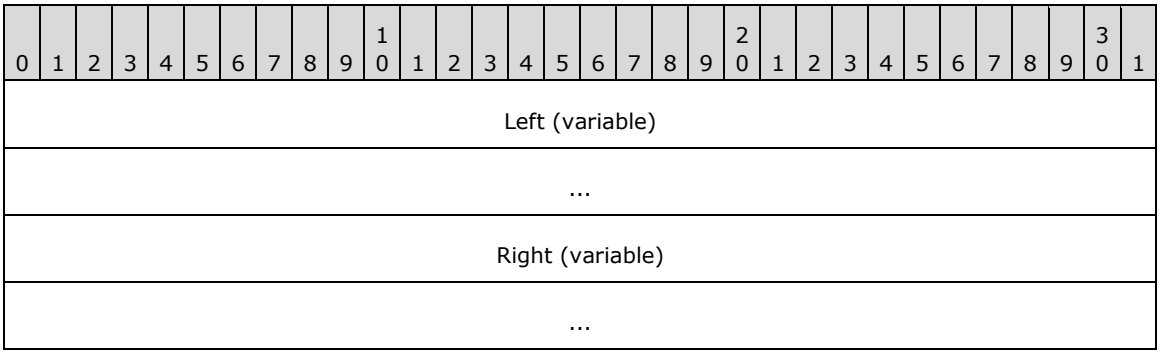
This object is generic and is used to specify different types of region node data, including:

- An EmfPlusRegionNodePath object, for a terminal node;
- An EmfPlusRectF object, for a terminal node; and
- An EmfPlusRegionNodeChildNodes object, for a non-terminal node.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.41 EmfPlusRegionNodeChildNodes Object

The EmfPlusRegionNodeChildNodes object specifies child nodes of a graphics region node.



Left (variable): An EmfPlusRegionNode object that specifies the left child node of this region node.

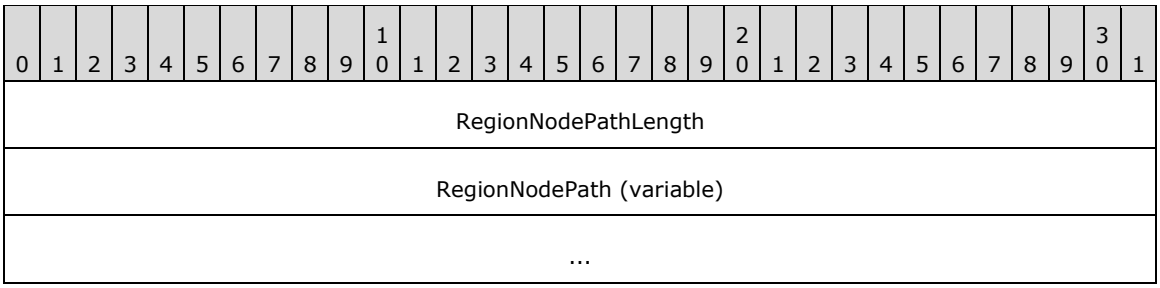
Right (variable): An EmfPlusRegionNode object that defines the right child node of this region node.

Graphics region nodes are specified with EmfPlusRegionNode objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.42 EmfPlusRegionNodePath Object

The EmfPlusRegionNodePath object specifies a graphics path for drawing the boundary of a region node.



RegionNodePathLength (4 bytes): A 32-bit signed integer that specifies the length in bytes of the **RegionNodePath** field.

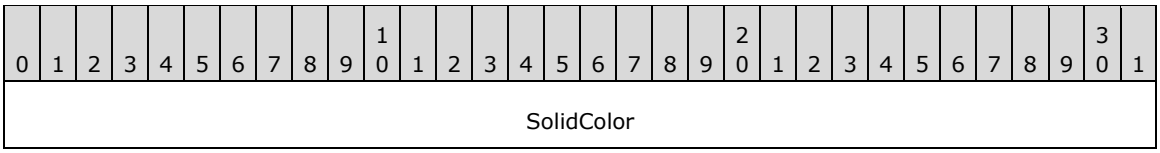
RegionNodePath (variable): An EmfPlusPath object that specifies the boundary of the region node.

Region nodes are specified by EmfPlusRegion objects. An EmfPlusRegionNodePath object MUST be present in the **RegionNodeData** field of an EmfPlusRegionNode object if its **Type** field is set to the **RegionNodeDataTypePath** value from the RegionNodeDataType enumeration.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.43 EmfPlusSolidBrushData Object

The EmfPlusSolidBrushData object specifies a solid color for a graphics brush.



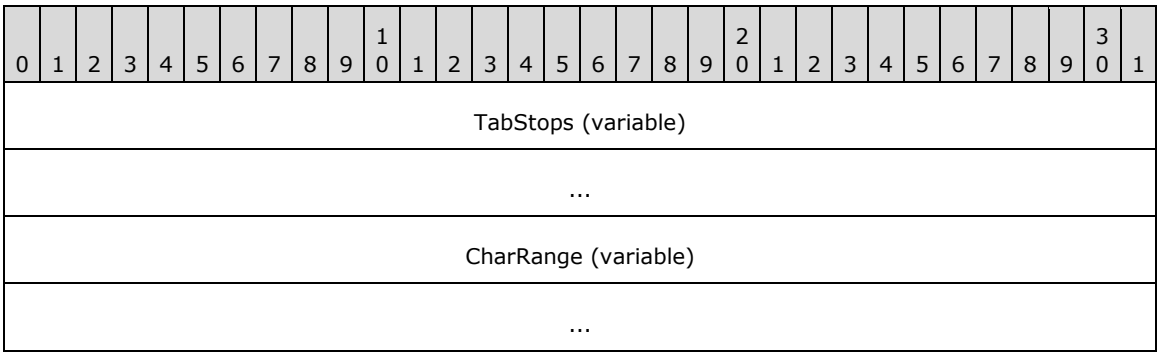
SolidColor (4 bytes): An EmfPlusARGB object that specifies the color of the brush.

Graphics brushes are specified by EmfPlusBrush objects. A solid color brush paints a background in a solid color.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.44 EmfPlusStringFormatData Object

The EmfPlusStringFormatData object specifies tab stops and character positions for a graphics string.



TabStops (variable): An optional array of floating-point values that specify the optional tab stop locations for this object. Each tab stop value represents the number of spaces between tab stops or, for the first tab stop, the number of spaces between the beginning of a line of text and the first tab stop.

This field MUST be present if the value of the **TabStopCount** field in the EmfPlusStringFormat object is greater than 0.

CharRange (variable): An optional array of **RangeCount** EmfPlusCharacterRange objects that specify the range of character positions within a string of text. The bounding region is defined by the area of the display that is occupied by a group of characters specified by the character range.

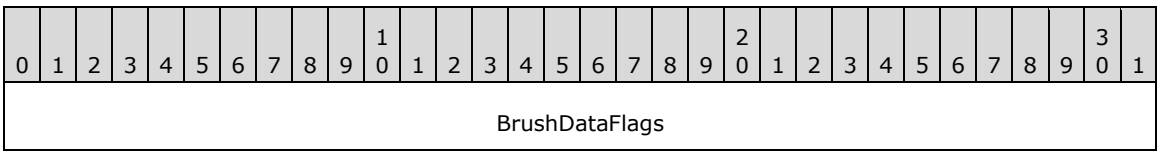
This field MUST be present if the value of the **RangeCount** field in the EmfPlusStringFormat object is greater than 0.

Graphics strings are specified by EmfPlusStringFormat objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.45 EmfPlusTextureBrushData Object

The EmfPlusTextureBrushData object specifies a texture image for a graphics brush.



WrapMode
OptionalData (variable)
...

BrushDataFlags (4 bytes): A 32-bit unsigned integer that specifies the data in the **OptionalData** field. This value MUST be composed of BrushData flags. The following flags are relevant to a texture brush:

Name	Value
BrushDataTransform	0x00000002
BrushDataIsGammaCorrected	0x00000080
BrushDataDoNotTransform	0x00000100

WrapMode (4 bytes): A 32-bit signed integer from the WrapMode enumeration that specifies how to repeat the texture image across a shape, when the image is smaller than the area being filled.

OptionalData (variable): An optional EmfPlusTextureBrushOptionalData object that specifies additional data for the texture brush. The specific contents of this field are determined by the value of the **BrushDataFlags** field.

Graphics brushes are specified by EmfPlusBrush objects. A texture brush paints an image, which in this context is called a "texture". The texture consists of either a portion of an image or a scaled version of an image, which is specified by an EmfPlusImage object in the **OptionalData** field.

Gamma correction controls the overall brightness and intensity of an image. Uncorrected images can look either bleached out or too dark. Varying the amount of gamma correction changes not only the brightness but also the ratios of red to green to blue. The need for gamma correction arises because an output device might not render colors in the same intensity as the input image.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.46 EmfPlusTextureBrushOptionalData Object

The EmfPlusTextureBrushOptionalData object specifies optional data for a texture brush.

Note: Each field of this object is optional and might not be present in the **OptionalData** field of an EmfPlusTextureBrushData object, depending on the BrushData flags set in its **BrushDataFlags** field. Although it is not practical to represent every possible combination of fields present or absent, this section specifies their relative order in the object. The implementer is responsible for determining which fields are actually present in a given metafile record, and for unmarshaling the data for individual fields separately and appropriately.

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
TransformMatrix (24 bytes, optional)																															
...																															
...																															

ImageObject (variable)
...

TransformMatrix (24 bytes): An optional EmfPlusTransformMatrix object that specifies a world space to device space transform for the texture brush. This field MUST be present if the **BrushDataTransform** flag is set in the **BrushDataFlags** field of the EmfPlusTextureBrushData object.

ImageObject (variable): An optional EmfPlusImage object that specifies the brush texture. This field MUST be present if the size of the EmfPlusObject record that defines this texture brush is large enough to accommodate an EmfPlusImage object in addition to the required fields of the EmfPlusTextureBrushData object and optionally an EmfPlusTransformMatrix object.

Graphics brushes are specified by EmfPlusBrush objects.

See section 2.2.2 for the specification of additional structure objects.

2.2.2.47 EmfPlusTransformMatrix Object

The EmfPlusTransformMatrix object specifies a world space to device space transform.

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
TransformMatrix (24 bytes)																															
...																															
...																															

TransformMatrix (24 bytes): This field specifies an **affine transform**, which requires a 2x2 matrix for a linear transformation and a 1x2 matrix for a translation. These values map to the coordinates of the transform matrix as follows:

- **TransformMatrix[0]** Corresponds to m11, which is the coordinate of the first row and first column of the 2x2 matrix.
- **TransformMatrix[1]** Corresponds to m12, which is the coordinate of the first row and second column of the 2x2 matrix.
- **TransformMatrix[2]** Corresponds to m21, which is the coordinate of the second row and first column of the 2x2 matrix.
- **TransformMatrix[3]** Corresponds to m22, which is the coordinate of the second row and second column of the 2x2 matrix.
- **TransformMatrix[4]** Corresponds to dx, which is the horizontal displacement in the 1x2 matrix.
- **TransformMatrix[5]** Corresponds to dy, which is the vertical displacement in the 1x2 matrix.

See section 2.2.2 for the specification of additional structure objects.

2.2.3 Image Effects Object Types

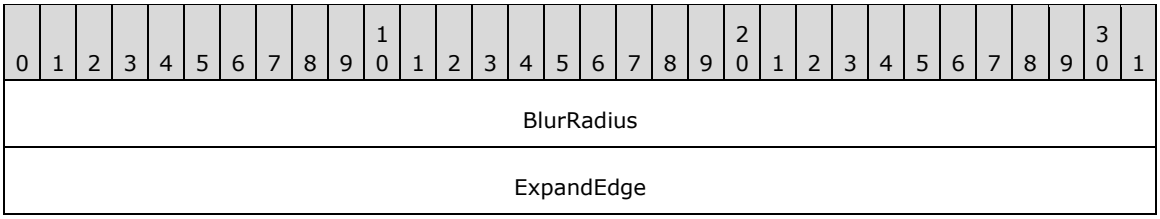
The Image Effects Objects specify parameters for graphics image effects, which can be applied to bitmap images.<16>

Parameters are specified for the following image effects:

Name	Section	Description
BlurEffect	2.2.3.1	Specifies a decrease in the difference in intensity between pixels in an image.
BrightnessContrastEffect	2.2.3.2	Specifies an expansion or contraction of the lightest and darkest areas of an image.
ColorBalanceEffect	2.2.3.3	Specifies adjustments to the relative amounts of red, green, and blue in an image.
ColorCurveEffect	2.2.3.4	Specifies one of eight adjustments to an image, including exposure, density, contrast, **highlight**, **shadow**, **midtone**, white saturation, or black saturation.
ColorLookupTableEffect	2.2.3.5	Specifies adjustments to the colors in an image.
ColorMatrixEffect	2.2.3.6	Specifies an affine transform to be applied to an image.
HueSaturationLightnessEffect	2.2.3.7	Specifies adjustments to the **hue**, **saturation**, and **lightness** of an image.
LevelsEffect	2.2.3.8	Specifies adjustments to the highlights, midtones, and shadows of an image.
RedEyeCorrectionEffect	2.2.3.9	Specifies areas of an image to which a red-eye correction effect is applied.
SharpenEffect	2.2.3.10	Specifies an increase in the difference in intensity between pixels in an image.
TintEffect	2.2.3.11	Specifies an addition of black or white to a specified hue in an image.

2.2.3.1 BlurEffect Object

The BlurEffect object specifies a decrease in the difference in intensity between pixels in an image.



BlurRadius (4 bytes): A 32-bit floating-point number that specifies the blur radius in pixels, which determines the number of pixels involved in calculating the new value of a given pixel. This value MUST be in the range 0.0 through 255.0.

As this value increases, the number of pixels involved in the calculation increases, and the resulting bitmap SHOULD become more blurry.

ExpandEdge (4 bytes): A 32-bit Boolean value that specifies whether the bitmap expands by an amount equal to the value of the **BlurRadius** to produce soft edges. This value MUST be one of the following:

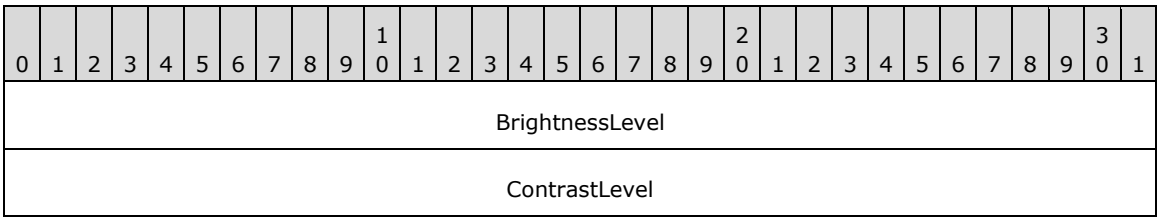
Value	Meaning
FALSE 0x00000000	The size of the bitmap MUST NOT change, and its soft edges SHOULD be clipped to the size of the **BlurRadius**.
TRUE 0x00000001	The size of the bitmap SHOULD expand by an amount equal to the **BlurRadius** to produce soft edges.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.2 BrightnessContrastEffect Object

The BrightnessContrastEffect object specifies an expansion or contraction of the lightest and darkest areas of an image.



BrightnessLevel (4 bytes): A 32-bit signed integer that specifies the brightness level. This value MUST be in the range -255 through 255, with effects as follows:

Value	Meaning
-255 ≤ *value* < 0	As the value decreases, the brightness of the image SHOULD decrease.
0	A value of 0 specifies that the brightness MUST NOT change.
0 < *value* ≤ 255	As the value increases, the brightness of the image SHOULD increase.

ContrastLevel (4 bytes): A 32-bit signed integer that specifies the contrast level. This value MUST be in the range -100 through 100, with effects as follows:

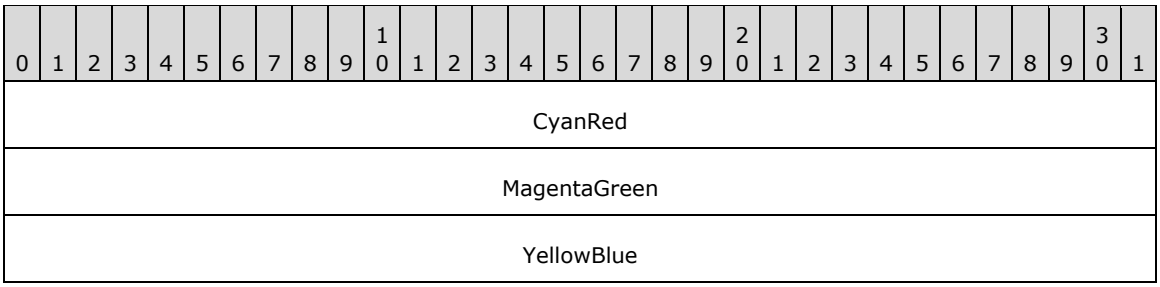
Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the contrast of the image SHOULD decrease.
0	A value of 0 specifies that the contrast MUST NOT change.
0 < *value* ≤ 100	As the value increases, the contrast of the image SHOULD increase.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.3 ColorBalanceEffect Object

The ColorBalanceEffect object specifies adjustments to the relative amounts of red, green, and blue in an image.



CyanRed (4 bytes): A 32-bit signed integer that specifies a change in the amount of red in the image. This value MUST be in the range -100 through 100, with effects as follows:

Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the amount of red in the image SHOULD decrease and the amount of cyan SHOULD increase.
0	A value of 0 specifies that the amounts of red and cyan MUST NOT change.
0 < *value* ≤ 100	As the value increases, the amount of red in the image SHOULD increase and the amount of cyan SHOULD decrease.

MagentaGreen (4 bytes): A 32-bit signed integer that specifies a change in the amount of green in the image. This value MUST be in the range -100 through 100, with effects as follows:

Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the amount of green in the image SHOULD decrease and the amount of magenta SHOULD increase.
0	A value of 0 specifies that the amounts of green and magenta MUST NOT change.
0 < *value* ≤ 100	As the value increases, the amount of green in the image SHOULD increase and the amount of magenta SHOULD decrease.

YellowBlue (4 bytes): A 32-bit signed integer that specifies a change in the amount of blue in the image. This value MUST be in the range -100 through 100, with effects as follows:

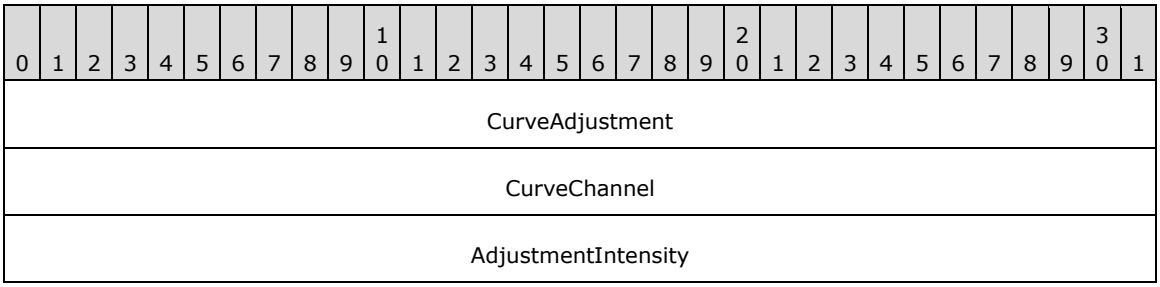
Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the amount of blue in the image SHOULD decrease and the amount of yellow SHOULD increase.
0	A value of 0 specifies that the amounts of blue and yellow MUST NOT change.
0 < *value* ≤ 100	As the value increases, the amount of blue in the image SHOULD increase and the amount of yellow SHOULD decrease.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.4 ColorCurveEffect Object

The ColorCurveEffect object specifies one of eight adjustments to the color curve of an image.



CurveAdjustment (4 bytes): A 32-bit unsigned integer that specifies the curve adjustment to apply to the colors in bitmap. This value MUST be defined in the CurveAdjustments enumeration.

CurveChannel (4 bytes): A 32-bit unsigned integer that specifies the color channel to which the curve adjustment applies. This value MUST be defined in the CurveChannel enumeration.

AdjustmentIntensity (4 bytes): A 32-bit signed integer that specifies the intensity of the curve adjustment to the color channel specified by **CurveChannel**. The ranges of meaningful values for this field vary according to the **CurveAdjustment** value, as follows:

Exposure adjustment range:

Value	Meaning
-255 ≤ *value* < 0	As the value decreases, the exposure of the image SHOULD decrease.
0	A value of 0 specifies that the exposure MUST NOT change.
0 < *value* ≤ 255	As the value increases, the exposure of the image SHOULD increase.

Density adjustment range:

Value	Meaning
-255 ≤ *value* < 0	As the value decreases, the density of the image SHOULD decrease, resulting in a darker image.
0	A value of 0 specifies that the density MUST NOT change.
0 < *value* ≤ 255	As the value increases, the density of the image SHOULD increase.

Contrast adjustment range:

Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the contrast of the image SHOULD decrease.
0	A value of 0 specifies that the contrast MUST NOT change.
0 < *value* ≤ 100	As the value increases, the contrast of the image SHOULD increase.

Highlight adjustment range:

Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the light areas of the image SHOULD appear darker.
0	A value of 0 specifies that the highlight MUST NOT change.
0 < *value* ≤ 100	As the value increases, the light areas of the image SHOULD appear lighter.

Shadow adjustment range:

Value	Meaning
-100 ≤ *value* < 0	As the value decreases, the dark areas of the image SHOULD appear darker.
0	A value of 0 specifies that the shadow MUST NOT change.
0 < *value* ≤ 100	As the value increases, the dark areas of the image SHOULD appear lighter.

White saturation adjustment range:

Value	Meaning
0 — 255	As the value increases, the upper limit of the range of color channel intensities increases.

Black saturation adjustment range:

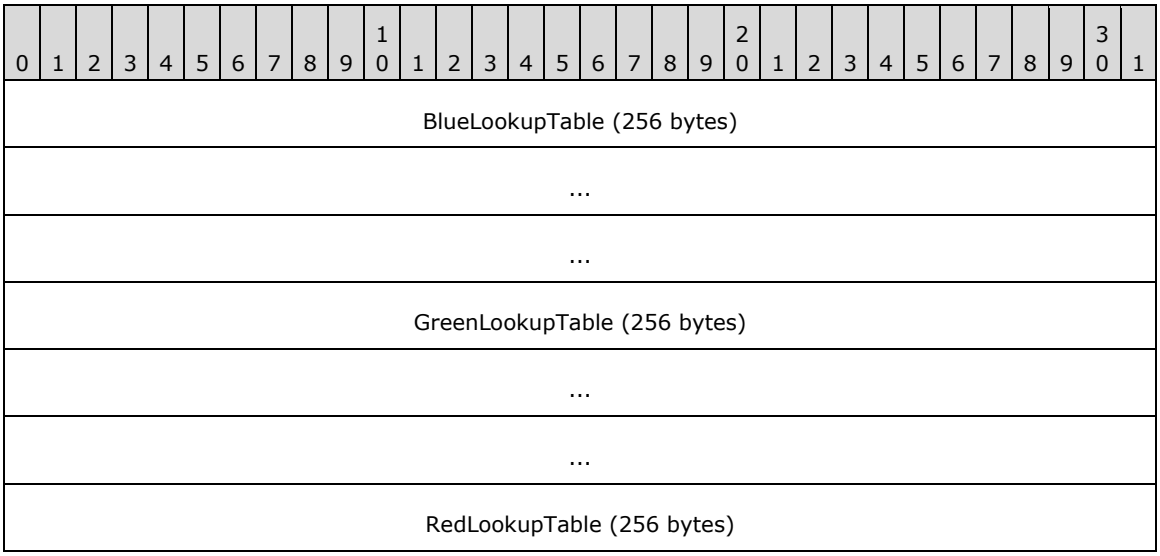
Value	Meaning
0 — 255	As the value increases, the lower limit of the range of color channel intensities increases.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.5 ColorLookupTableEffect Object

The ColorLookupTableEffect object specifies adjustments to the colors in an image.



...
...
AlphaLookupTable (256 bytes)
...
...

BlueLookupTable (256 bytes): An array of 256 bytes that specifies the adjustment for the blue color channel.

GreenLookupTable (256 bytes): An array of 256 bytes that specifies the adjustment for the green color channel.

RedLookupTable (256 bytes): An array of 256 bytes that specifies the adjustment for the red color channel.

AlphaLookupTable (256 bytes): An array of 256 bytes that specifies the adjustment for the alpha color channel.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.6 ColorMatrixEffect Object

The ColorMatrixEffect object specifies an affine transform to be applied to an 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
Matrix_N_0 (20 bytes)																															
...																															
...																															
Matrix_N_1 (20 bytes)																															
...																															
...																															
Matrix_N_2 (20 bytes)																															
...																															
...																															
Matrix_N_3 (20 bytes)																															

...
...
Matrix_N_4 (20 bytes)
...
...

Matrix_N_0 (20 bytes): Matrix[N][0] of the 5x5 **color matrix**. This row is used for transforms.

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
Matrix_0_0																															
Matrix_1_0																															
Matrix_2_0																															
Matrix_3_0																															
Matrix_4_0																															

Matrix_0_0 (4 bytes): Matrix[0][0], which is the factor for the color red.

Matrix_1_0 (4 bytes): Matrix[1][0].

Matrix_2_0 (4 bytes): Matrix[2][0].

Matrix_3_0 (4 bytes): Matrix[3][0].

Matrix_4_0 (4 bytes): Matrix[4][0]. This value MUST be 0.0.

Matrix_N_1 (20 bytes): Matrix[N][1] of the 5x5 color matrix. This row is used for transforms.

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
Matrix_0_1																															
Matrix_1_1																															
Matrix_2_1																															
Matrix_3_1																															
Matrix_4_1																															

Matrix_0_1 (4 bytes): Matrix[0][1].

Matrix_1_1 (4 bytes): Matrix[1][1], which is the factor for the color green.

Matrix_2_1 (4 bytes): Matrix[2][1].

Matrix_3_1 (4 bytes): Matrix[3][1].

Matrix_4_1 (4 bytes): Matrix[4][1]. This value MUST be 0.0.

Matrix_N_2 (20 bytes): Matrix[N][2] of the 5x5 color matrix. This row is used for transforms.

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
Matrix_0_2																															
Matrix_1_2																															
Matrix_2_2																															
Matrix_3_2																															
Matrix_4_2																															

Matrix_0_2 (4 bytes): Matrix[0][2].

Matrix_1_2 (4 bytes): Matrix[1][2].

Matrix_2_2 (4 bytes): Matrix[2][2], which is the factor for the color blue.

Matrix_3_2 (4 bytes): Matrix[3][2].

Matrix_4_2 (4 bytes): Matrix[4][2]. This value MUST be 0.0.

Matrix_N_3 (20 bytes): Matrix[N][3] of the 5x5 color matrix. This row is used for transforms.

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
Matrix_0_3																															
Matrix_1_3																															
Matrix_2_3																															
Matrix_3_3																															
Matrix_4_3																															

Matrix_0_3 (4 bytes): Matrix[0][3].

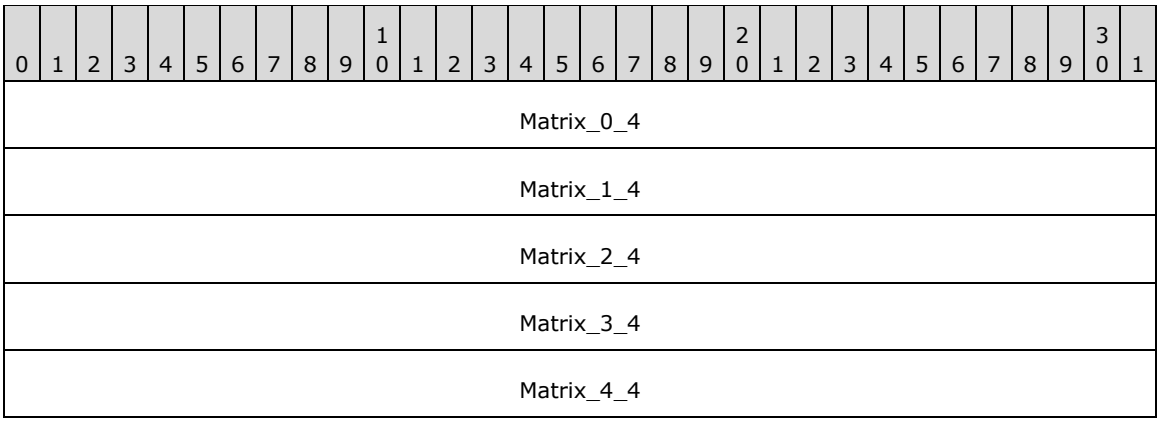
Matrix_1_3 (4 bytes): Matrix[1][3].

Matrix_2_3 (4 bytes): Matrix[2][3].

Matrix_3_3 (4 bytes): Matrix[3][3], which is the factor for the alpha (transparency) value.

Matrix_4_3 (4 bytes): Matrix[4][3]. This value MUST be 0.0.

Matrix_N_4 (20 bytes): Matrix[N][4] of the 5x5 color matrix. This row is used for color translations.



Matrix_0_4 (4 bytes): Matrix[0][4].

Matrix_1_4 (4 bytes): Matrix[1][4].

Matrix_2_4 (4 bytes): Matrix[2][4].

Matrix_3_4 (4 bytes): Matrix[3][4].

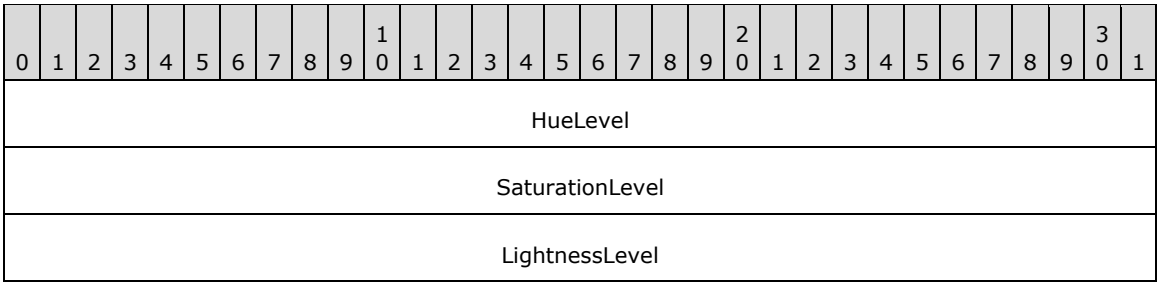
Matrix_4_4 (4 bytes): Matrix[4][4]. This value SHOULD be 1.0.<17>

Bitmap images are specified by EmfPlusBitmap objects. A color matrix effect is performed by multiplying a **color vector** by a ColorMatrixEffect object. A 5x5 color matrix can perform a linear transform, including **reflection**, **rotation**, **shearing**, or scaling followed by a translation.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.7 HueSaturationLightnessEffect Object

The HueSaturationLightnessEffect object specifies adjustments to the hue, saturation, and lightness of an image.



HueLevel (4 bytes): Specifies the adjustment to the hue.

Value	Meaning
-180 ≤ *value* < 0	Negative values specify clockwise rotation on the **color wheel**.
0	A value of 0 specifies that the hue MUST NOT change.
0 < *value* ≤ 180	Positive values specify counter-clockwise rotation on the color wheel.

SaturationLevel (4 bytes): Specifies the adjustment to the saturation.

Value	Meaning
-100 ≤ *value* < 0	Negative values specify decreasing saturation.
0	A value of 0 specifies that the saturation MUST NOT change.
0 < *value* ≤ 100	Positive values specify increasing saturation.

LightnessLevel (4 bytes): Specifies the adjustment to the lightness.

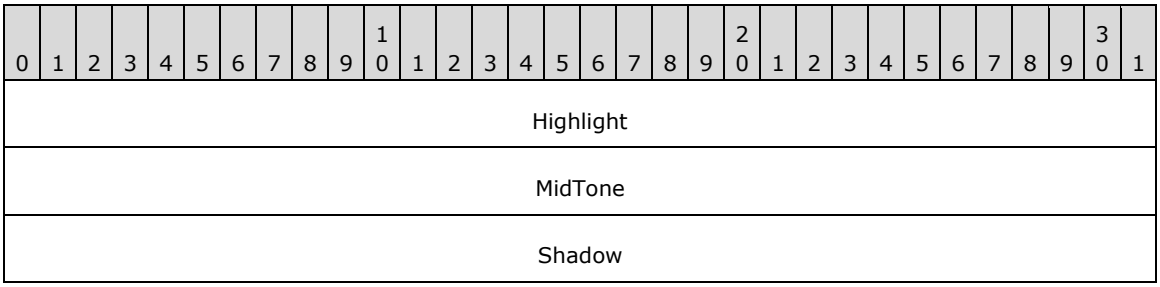
Value	Meaning
-100 ≤ *value* < 0	Negative values specify decreasing lightness.
0	A value of 0 specifies that the lightness MUST NOT change.
0 < *value* ≤ 100	Positive values specify increasing lightness.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.8 LevelsEffect Object

The LevelsEffect object specifies adjustments to the highlights, midtones, and shadows of an image.



Highlight (4 bytes): Specifies how much to lighten the highlights of an image. The color channel values at the high end of the intensity range are altered more than values near the middle or low ends, which means an image can be lightened without losing the contrast between the darker portions of the image.

Value	Meaning
0 ≤ *value* < 100	Specifies that highlights with a percent of intensity above this threshold SHOULD be increased.
100	Specifies that highlights MUST NOT change.

MidTone (4 bytes): Specifies how much to lighten or darken the midtones of an image. Color channel values in the middle of the intensity range are altered more than values near the high or low ends, which means an image can be lightened or darkened without losing the contrast between the darkest and lightest portions of the image.

Value	Meaning
-100 ≤ *value* < 0	Specifies that midtones are made darker.
0	Specifies that midtones MUST NOT change.

Value	Meaning
0 < *value* ≤ 100	Specifies that midtones are made lighter.

Shadow (4 bytes): Specifies how much to darken the shadows of an image. Color channel values at the low end of the intensity range are altered more than values near the middle or high ends, which means an image can be darkened without losing the contrast between the lighter portions of the image.

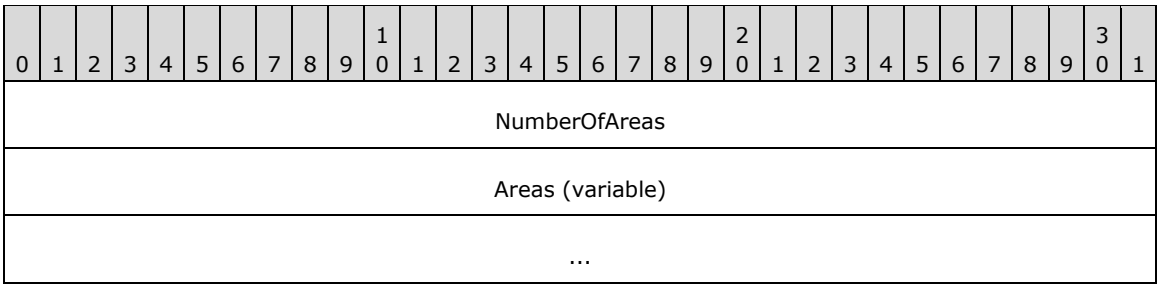
Value	Meaning
0	Specifies that shadows MUST NOT change.
0 < *value* ≤ 100	Specifies that shadows with a percent of intensity below this threshold are made darker.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.9 RedEyeCorrectionEffect Object

The RedEyeCorrectionEffect object specifies areas of an image to which a red-eye correction is applied.



NumberOfAreas (4 bytes): A 32-bit signed integer that specifies the number of rectangles in the **Areas** field.

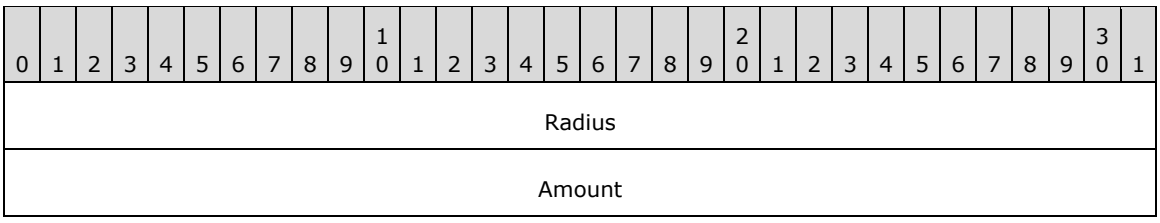
Areas (variable): An array of **NumberOfAreas** WMF RectL objects [MS-WMF]. Each rectangle specifies an area of the bitmap image to which the red-eye correction effect SHOULD be applied.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.10 SharpenEffect Object

The SharpenEffect object specifies an increase in the difference in intensity between pixels in an image.



Radius (4 bytes): A 32-bit floating-point number that specifies the sharpening radius in pixels, which determines the number of pixels involved in calculating the new value of a given pixel.

As this value increases, the number of pixels involved in the calculation increases, and the resulting bitmap SHOULD become sharper.

Amount (4 bytes): A 32-bit floating-point number that specifies the difference in intensity between a given pixel and the surrounding pixels.

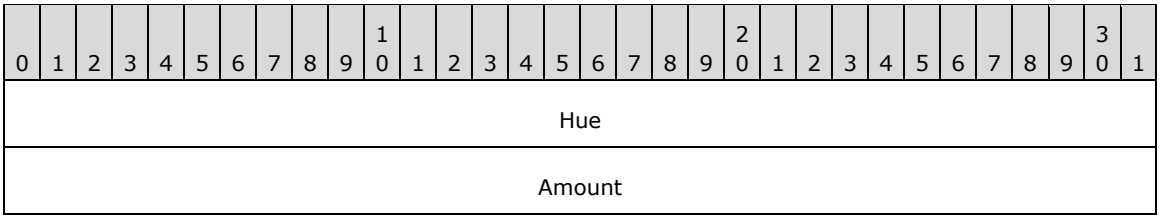
Value	Meaning
0	Specifies that sharpening MUST NOT be performed.
0 < *value* ≤ 100	As this value increases, the difference in intensity between pixels SHOULD increase.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.2.3.11 TintEffect Object

The TintEffect object specifies an addition of black or white to a specified hue in an image.



Hue (4 bytes): A 32-bit signed integer that specifies the hue to which the tint effect is applied.

Value	Meaning
-180 ≤ *value* < 0	The color at a specified counter-clockwise rotation of the color wheel, starting from blue.
0	A value of 0 specifies the color blue on the color wheel.
0 < *value* ≤ 180	The color at a specified clockwise rotation of the color wheel, starting from blue.

Amount (4 bytes): A 32-bit signed integer that specifies how much the hue is strengthened or weakened.

Value	Meaning
-100 ≤ *value* < 0	Negative values specify how much the hue is weakened, which equates to the addition of black.
0	A value of 0 specifies that the **tint** MUST NOT change.
0 < *value* ≤ 100	Positive values specify how much the hue is strengthened, which equates to the addition of white.

Bitmap images are specified by EmfPlusBitmap objects.

See section 2.2.3 for the specification of additional image effects parameter objects.

2.3 EMF+ Records

This section specifies the Records, which are grouped into the following categories:

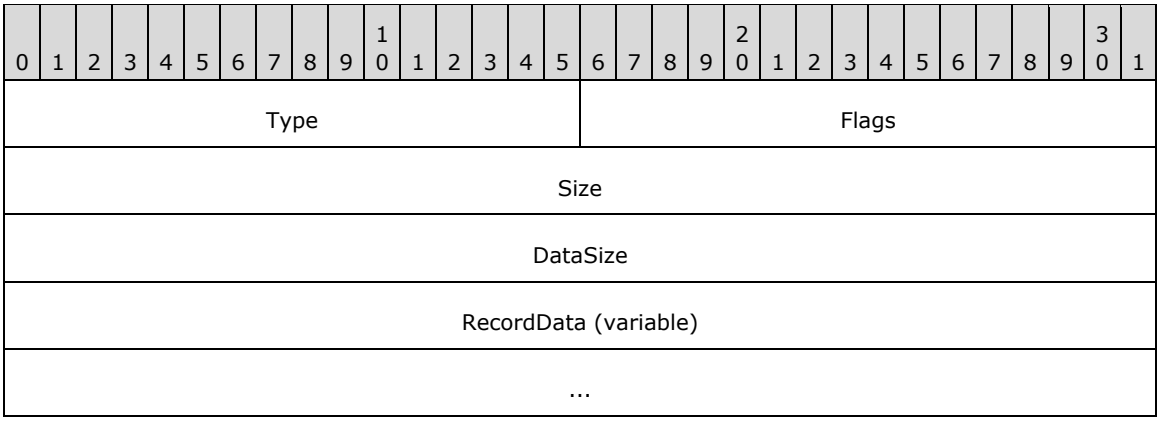
Name	Section	Description
Clipping record types	2.3.1	Specify clipping regions and operations.
Comment record types	2.3.2	Specify arbitrary private data in the EMF+ metafile.
Control record types	2.3.3	Specify global parameters for EMF+ metafile processing.
Drawing record types	2.3.4	Specify graphics output.
Object record types	2.3.5	Define reusable graphics objects.
Property record types	2.3.6	Specify properties of the playback device context.
State record types	2.3.7	Specify operations on the state of the playback device context.
Terminal Server record types	2.3.8	Specify graphics processing on a terminal server.
Transform record types	2.3.9	Specify properties and transforms on coordinate spaces.

2.3.1 Clipping Record Types

The clipping record types specify clipping regions and operations. The following are EMF+ clipping record types:

Name	Section	Description
EmfPlusOffsetClip	2.3.1.1	Applies a translation transform on the current clipping region for the world space.
EmfPlusResetClip	2.3.1.2	Resets the current clipping region for the world space to infinity.
EmfPlusSetClipPath	2.3.1.3	Combines the current clipping region with a graphics path.
EmfPlusSetClipRect	2.3.1.4	Combines the current clipping region with a rectangle.
EmfPlusSetClipRegion	2.3.1.5	Combines the current clipping region with another graphics region.

The generic structure of EMF+ clipping records is specified as follows:



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The clipping record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusResetClip	0x4031
EmfPlusSetClipRect	0x4032
EmfPlusSetClipPath	0x4033
EmfPlusSetClipRegion	0x4034
EmfPlusOffsetClip	0x4035

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

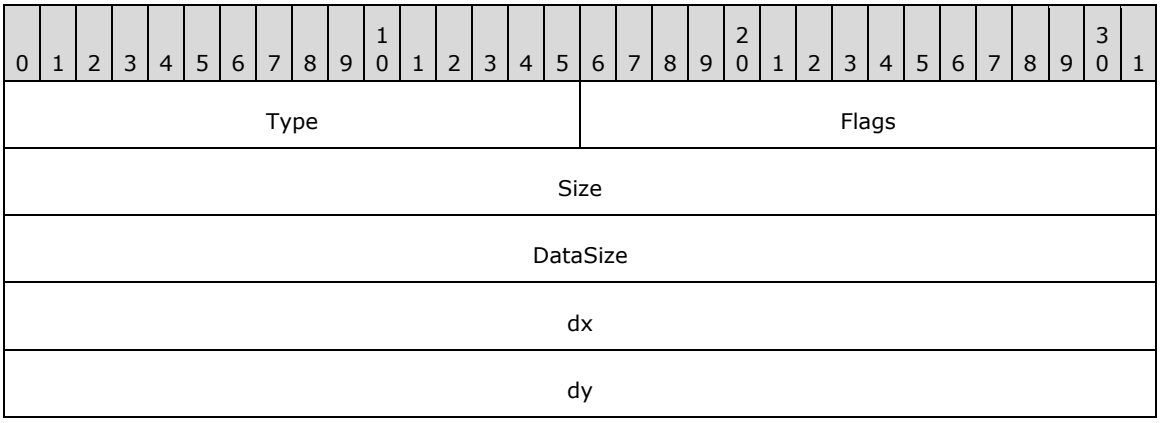
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit–aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit–aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, defines the data specific to individual records. For specifications of the additional information, if any, contained within this field, see individual record definitions.

2.3.1.1 EmfPlusOffsetClip Record

The EmfPlusOffsetClip record applies a translation transform on the current clipping region for the world space.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusOffsetClip from the RecordType enumeration. The value MUST be 0x4035.

Flags (2 bytes): A 16-bit unsigned integer that is reserved and MUST be ignored.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000014.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value MUST be 0x00000008.

dx (4 bytes): A 32-bit floating-point value that specifies the horizontal offset for the translation.

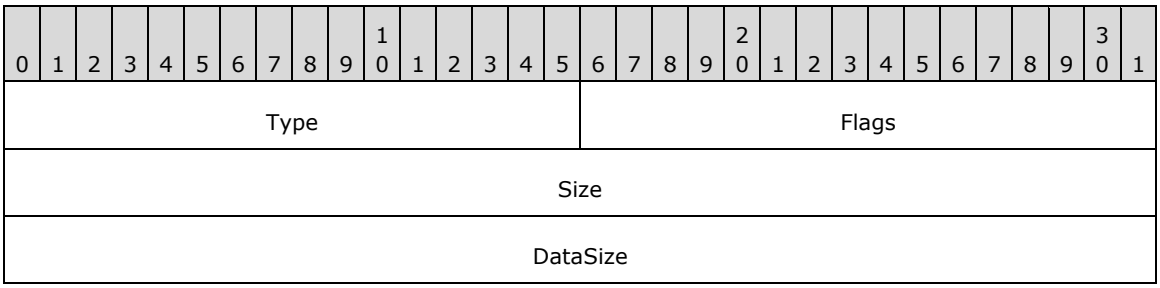
dy (4 bytes): A 32-bit floating-point value that specifies the vertical offset for the translation.

The new current clipping region is set to the result of the translation transform.

See section 2.3.1 for the specification of additional clipping record types.

2.3.1.2 EmfPlusResetClip Record

The EmfPlusResetClip record resets the current clipping region for the world space to infinity.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusResetClip from the RecordType enumeration. The value MUST be 0x4031.

Flags (2 bytes): A 16-bit unsigned integer that is reserved and MUST be ignored.

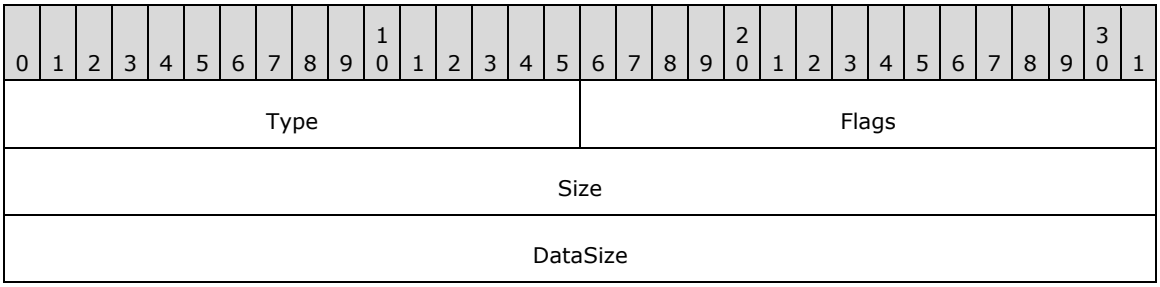
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value MUST be 0x00000000.

See section 2.3.1 for the specification of additional clipping record types.

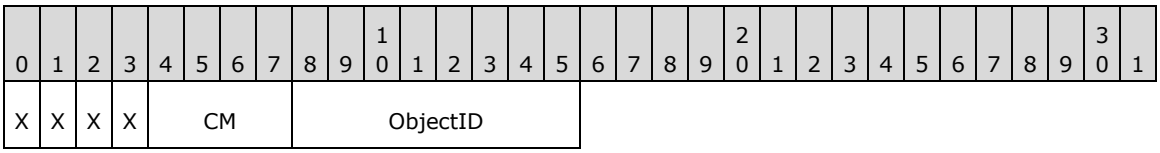
2.3.1.3 EmfPlusSetClipPath Record

The EmfPlusSetClipPath record combines the current clipping region with a graphics path.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetClipPath from the RecordType enumeration. The value MUST be 0x4033.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

CM (4 bits): Specifies the logical operation for combining two regions. See the CombineMode enumeration for the meanings of the values.

ObjectID (1 byte): The index of an EmfPlusPath object in the EMF+ Object Table (section 3.1.2). The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

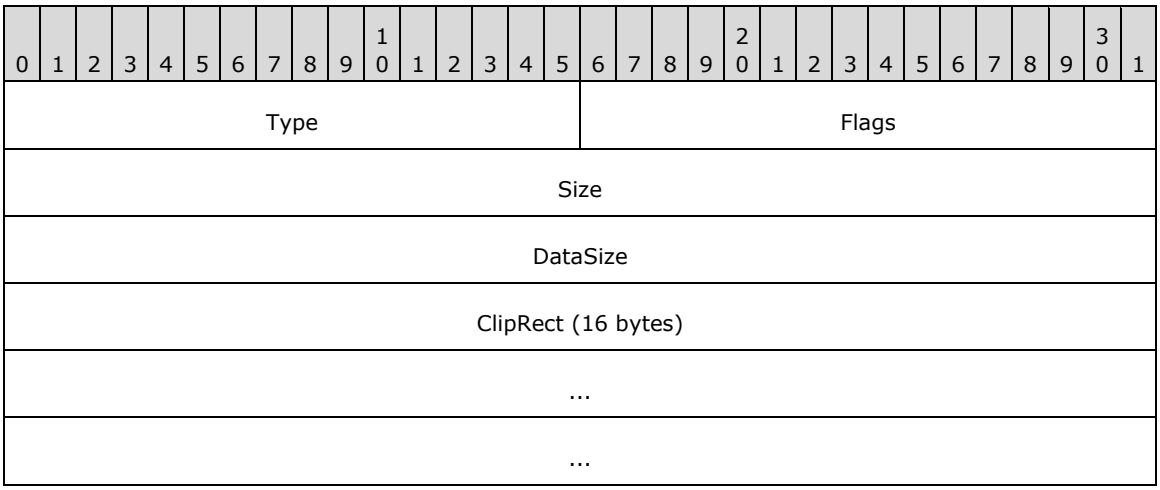
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value MUST be 0x00000000.

The new current clipping region is set to the result of the CombineMode operation.

See section 2.3.1 for the specification of additional clipping record types.

2.3.1.4 EmfPlusSetClipRect Record

The EmfPlusSetClipRect record combines the current clipping region with a rectangle.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetClipRect from the RecordType enumeration. The value MUST be 0x4032.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of 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
X	X	X	X	CM				X	X	X	X	X	X	X	X																

X (1 bit): Reserved and MUST be ignored.

CM (4 bits): Specifies the logical operation for combining two regions. See the CombineMode enumeration for the meanings of the values.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000001C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value MUST be 0x00000010.

ClipRect (16 bytes): An EmfPlusRectF object that defines the rectangle to use in the CombineMode operation.

The new current clipping region is set to the result of the CombineMode operation.

See section 2.3.1 for the specification of additional clipping record types.

2.3.1.5 EmfPlusSetClipRegion Record

The EmfPlusSetClipRegion record combines the current clipping region with another graphics region.

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
Type																Flags															
Size																															
DataSize																															

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as **EmfPlusSetClipRegion** from the RecordType enumeration. The value MUST be 0x4034.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of 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
X	X	X	X	CM				ObjectID																							

X (1 bit): Reserved and MUST be ignored.

CM (4 bits): Specifies the logical operation for combining two regions. See the CombineMode enumeration for the meanings of the values.

ObjectID (1 byte): The index of an EmfPlusRegion object in the EMF+ Object Table. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value MUST be 0x00000000.

The new current clipping region is set to the result of performing the CombineMode operation on the previous current clipping region and the specified EmfPlusRegion object.

See section 2.3.1 for the specification of additional clipping record types.

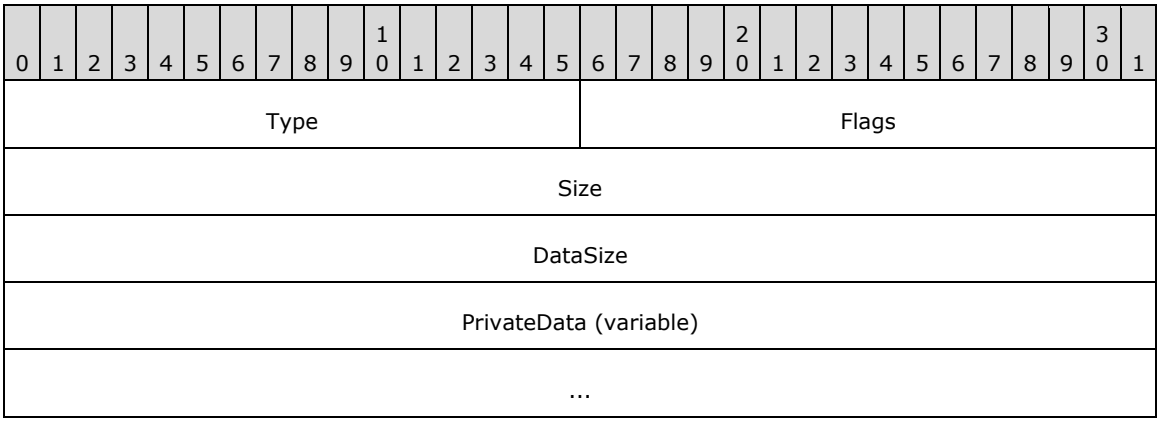
2.3.2 Comment Record Types

The Comment record type defines a format for specifying arbitrary private data.

Name	Section	Description
EmfPlusComment	2.3.2.1	Specifies arbitrary private data.

2.3.2.1 EmfPlusComment Record

The EmfPlusComment record specifies arbitrary private data.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusComment from the RecordType enumeration. The value MUST be 0x4003.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.<18>

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit–aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, it MUST be computed as follows:

```
Size = DataSize + 0x0000000C
```

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit–aligned number of bytes of record-specific data that follows.

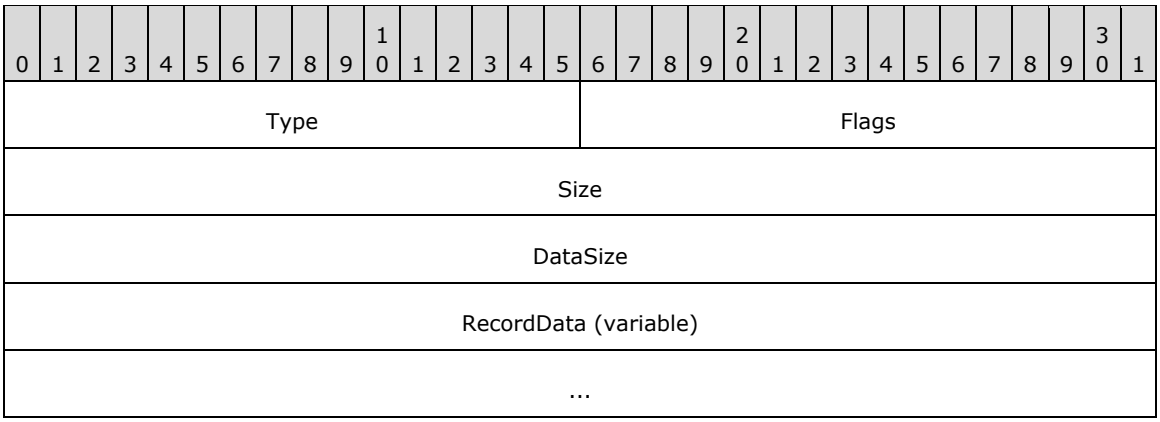
PrivateData (variable): A **DataSize**-length byte array of private data.

2.3.3 Control Record Types

The control record types specify global parameters for EMF+ metafile processing. The following are EMF+ control record types:

Name	Section	Description
EmfPlusEndOfFile	2.3.3.1	Specifies the end of EMF+ data in the metafile.
EmfPlusGetDC	2.3.3.2	Specifies that subsequent EMF records ([MS-EMF] section 2.3) encountered in the metafile SHOULD be processed.
EmfPlusHeader	2.3.3.3	Specifies the start of EMF+ data in the metafile.

The generic structure of EMF+ control records is specified as follows:



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The control record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusHeader	0x4001
EmfPlusEndOfFile	0x4002
EmfPlusGetDC	0x4004

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

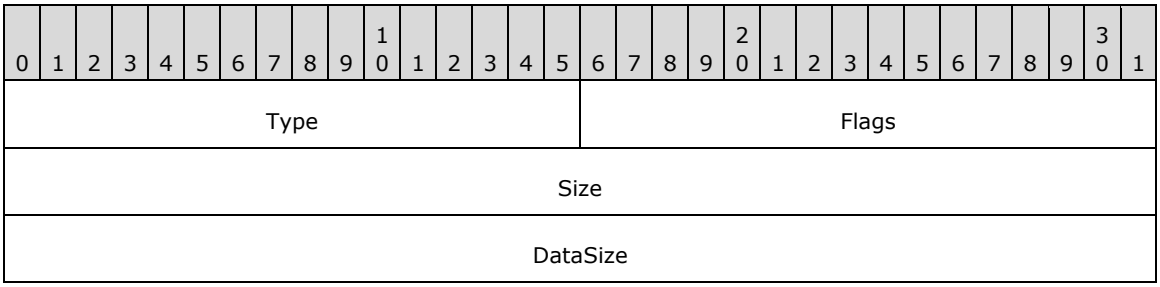
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit–aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit–aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, defines the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

2.3.3.1 EmfPlusEndOfFile Record

The EmfPlusEndOfFile record specifies the end of EMF+ data in the metafile.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusEndOfFile from the RecordType enumeration. The value MUST be 0x4002.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

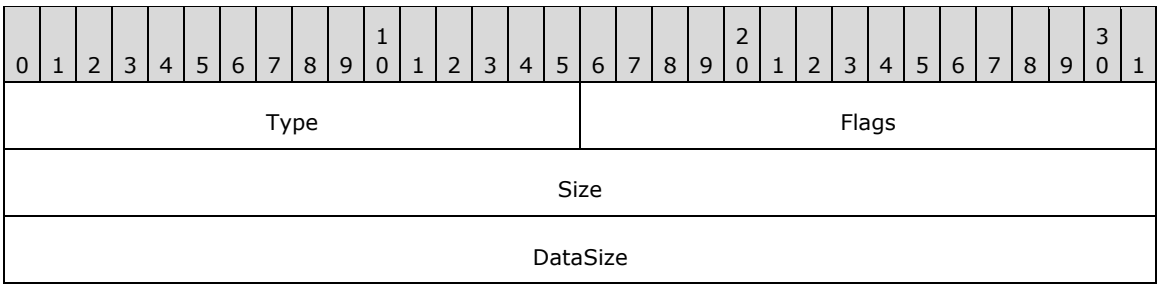
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.3 for the specification of additional control record types.

2.3.3.2 EmfPlusGetDC Record

The EmfPlusGetDC record specifies that subsequent EMF records ([MS-EMF] section 2.3) encountered in the metafile SHOULD be processed.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusGetDC from the RecordType enumeration. The value MUST be 0x4004.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data. For this record type, the value is 0x0000000C.

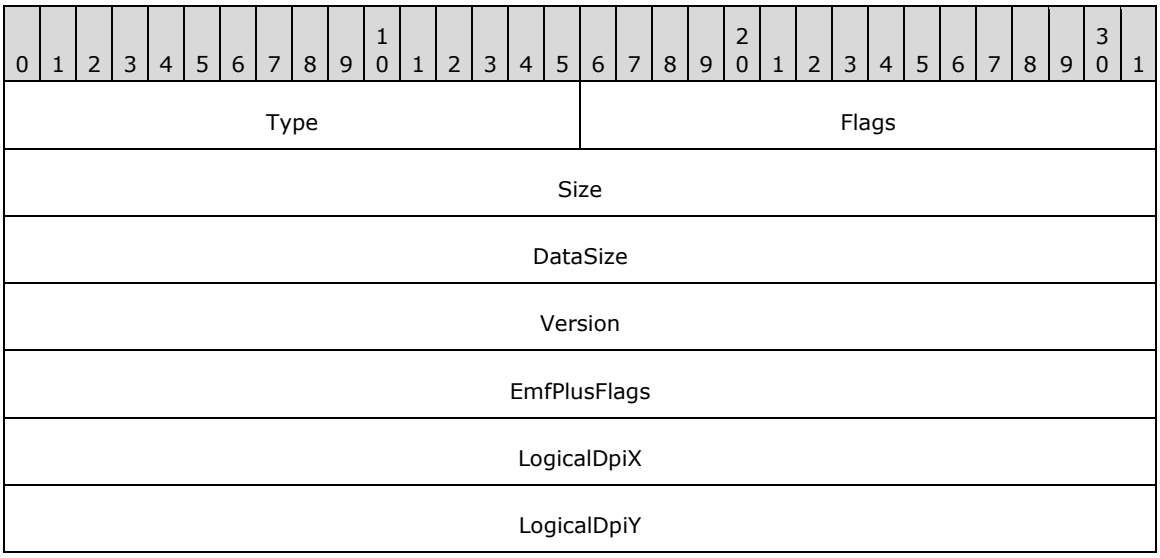
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value is 0x00000000.

EMF records cease being processed when the next EMF+ record is encountered.

See section 2.3.3 for the specification of additional control record types.

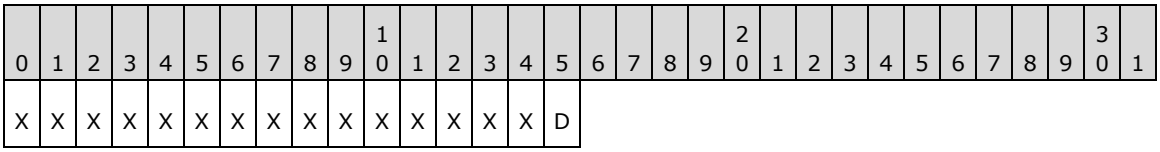
2.3.3.3 EmfPlusHeader Record

The EmfPlusHeader record specifies the start of EMF+ data in the metafile.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusHeader from the RecordType enumeration. The value MUST be 0x4001.

Flags (2 bytes): A 16-bit unsigned integer that provides information about the structure of the metafile.



X (1 bit): Reserved and MUST be ignored.

D (1 bit): If set, this flag indicates that this metafile is **EMF+ Dual**, which means that it contains two sets of records, each of which completely specifies the graphics content. If clear, the graphics content is specified by EMF+ records, and possibly EMF records ([MS-EMF] section 2.3) that are preceded by an EmfPlusGetDC record. If this flag is set, EMF records alone SHOULD suffice to define the graphics content. Note that whether the **EMF+ Dual** flag is set or not, some EMF records are always present, namely EMF control records and the EMF records that contain EMF+ records.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value is 0x0000001C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value is 0x00000010.

Version (4 bytes): An EmfPlusGraphicsVersion object that specifies the version of operating system graphics that was used to create this metafile.

EmfPlusFlags (4 bytes): A 32-bit unsigned integer that contains information about how this metafile was recorded.

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
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	V

X (1 bit): Reserved and MUST be ignored.

V (1 bit): If set, this flag indicates that the metafile was recorded with a reference device context for a video display. If clear, the metafile was recorded with a reference device context for a printer.

LogicalDpiX (4 bytes): A 32-bit unsigned integer that specifies the horizontal resolution for which the metafile was recorded, in units of pixels per inch.

LogicalDpiY (4 bytes): A 32-bit unsigned integer that specifies the vertical resolution for which the metafile was recorded, in units of lines per inch.

The EmfPlusHeader record MUST be embedded in an EMF EMR_COMMENT_EMFPLUS record, which MUST be the record immediately following the EMF header in the metafile.

See section 2.3.3 for the specification of additional control record types.

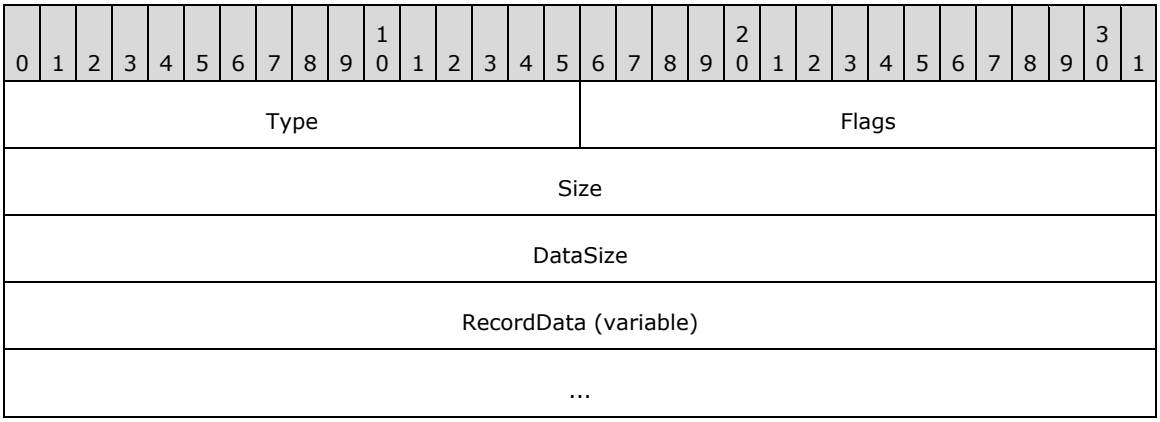
2.3.4 Drawing Record Types

The drawing record types specify graphics output. The following are EMF+ drawing record types:

Name	Section	Description
EmfPlusClear	2.3.4.1	Clears the output coordinate space and initializes it with a background color and transparency.
EmfPlusDrawArc	2.3.4.2	Specifies drawing the arc of an ellipse.
EmfPlusDrawBeziers	2.3.4.3	Specifies drawing a sequence of connected Bezier curves.
EmfPlusDrawClosedCurve	2.3.4.4	Specifies drawing a closed cardinal spline.
EmfPlusDrawCurve	2.3.4.5	Specifies drawing a cardinal spline.
EmfPlusDrawDriverString	2.3.4.6	Specifies text output with character positions.
EmfPlusDrawEllipse	2.3.4.7	Specifies drawing an ellipse.
EmfPlusDrawImage	2.3.4.8	Specifies drawing a scaled image.
EmfPlusDrawImagePoints	2.3.4.9	Specifies drawing a scaled image inside a parallelogram.
EmfPlusDrawLines	2.3.4.10	Specifies drawing a series of connected lines.
EmfPlusDrawPath	2.3.4.11	Specifies drawing a graphics path.
EmfPlusDrawPie	2.3.4.12	Specifies drawing a section of the interior of an ellipse.
EmfPlusDrawRects	2.3.4.13	Specifies drawing a series of rectangles.

Name	Section	Description
EmfPlusDrawString	2.3.4.14	Specifies text output with string formatting.
EmfPlusFillClosedCurve	2.3.4.15	Specifies filling the interior of a closed cardinal spline.
EmfPlusFillEllipse	2.3.4.16	Specifies filling the interior of an ellipse.
EmfPlusFillPath	2.3.4.17	Specifies filling the interior of a graphics path.
EmfPlusFillPie	2.3.4.18	Specifies filling a section of the interior of an ellipse.
EmfPlusFillPolygon	2.3.4.19	Specifies filling the interior of a polygon.
EmfPlusFillRects	2.3.4.20	Specifies filling the interiors of a series of rectangles.
EmfPlusFillRegion	2.3.4.21	Specifies filling the interior of a graphics region.

The generic structure of EMF+ drawing records is specified as follows:



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The drawing record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusClear	0x4009
EmfPlusFillRects	0x400A
EmfPlusDrawRects	0x400B
EmfPlusFillPolygon	0x400C
EmfPlusDrawLines	0x400D
EmfPlusFillEllipse	0x400E
EmfPlusDrawEllipse	0x400F
EmfPlusFillPie	0x4010
EmfPlusDrawPie	0x4011
EmfPlusDrawArc	0x4012
EmfPlusFillRegion	0x4013

Name	Value
EmfPlusFillPath	0x4014
EmfPlusDrawPath	0x4015
EmfPlusFillClosedCurve	0x4016
EmfPlusDrawClosedCurve	0x4017
EmfPlusDrawCurve	0x4018
EmfPlusDrawBeziers	0x4019
EmfPlusDrawImage	0x401A
EmfPlusDrawImagePoints	0x401B
EmfPlusDrawString	0x401C
EmfPlusDrawDriverString	0x4036

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

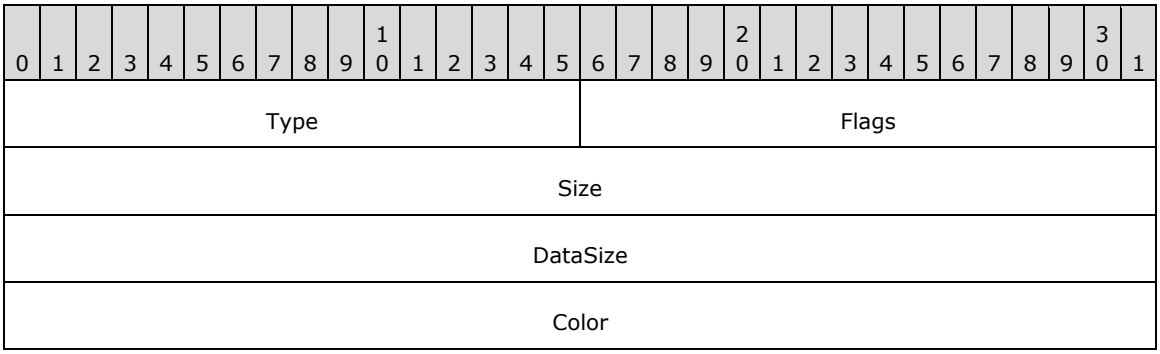
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, MUST define the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

2.3.4.1 EmfPlusClear Record

The EmfPlusClear record clears the output coordinate space and initializes it with a background color and transparency.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusClear from the RecordType enumeration. The value MUST be 0x4009.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, it MUST be 0x00000010.

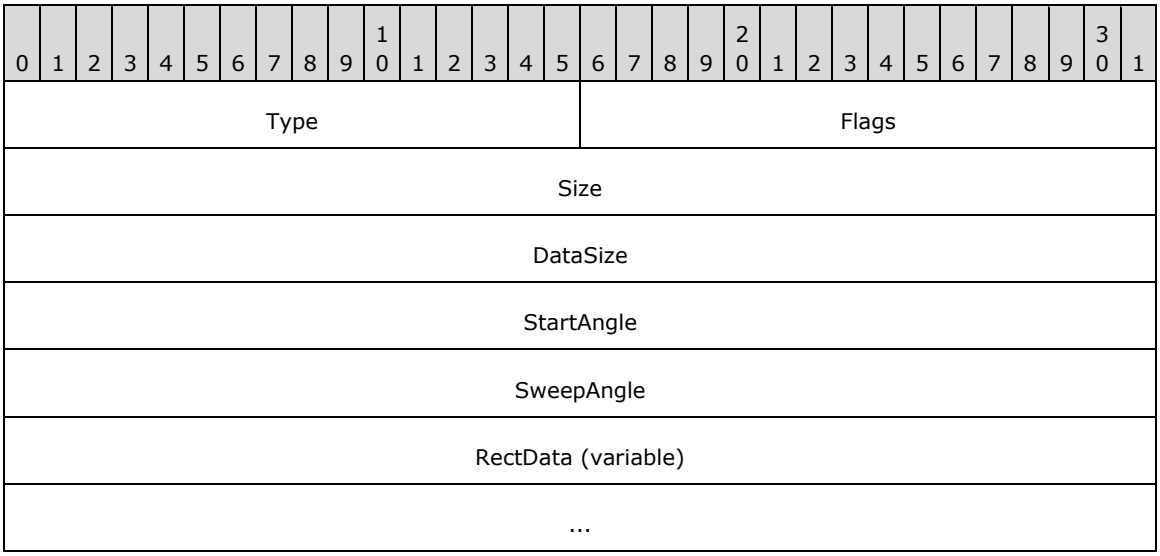
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, it MUST be 0x00000004.

Color (4 bytes): An EmfPlusARGB object that defines the color to paint the screen. All colors are specified in [IEC-RGB], unless otherwise noted.

See section 2.3.4 for the specification of additional drawing record types.

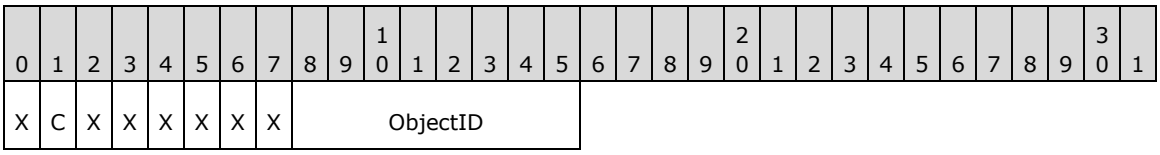
2.3.4.2 EmfPlusDrawArc Record

The EmfPlusDrawArc record specifies drawing the arc of an ellipse.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawArc from the RecordType enumeration. The value MUST be 0x4012.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the arc. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be one of the following:

Value	Meaning
0x0000001C	If the **C** bit is set in the **Flags** field.
0x00000024	If the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000010	If the **C** bit is set in the **Flags** field.
0x00000018	If the **C** bit is clear in the **Flags** field.

StartAngle (4 bytes): A 32-bit non-negative floating-point value that specifies the angle between the x-axis and the starting point of the arc. Any value is acceptable, but it MUST be interpreted modulo 360, with the result that is used being in the range 0.0 inclusive to 360.0 exclusive.

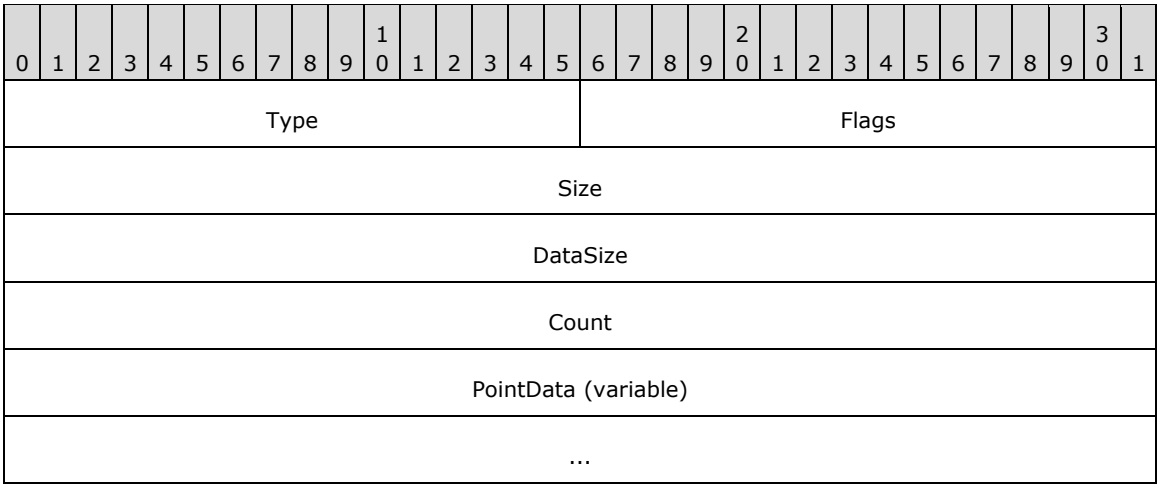
SweepAngle (4 bytes): A 32-bit floating-point value that specifies the extent of the arc to draw, as an angle in degrees measured from the starting point defined by the **StartAngle** value. Any value is acceptable, but it MUST be clamped to -360.0 to 360.0 inclusive. A positive value indicates that the sweep is defined in a clockwise direction, and a negative value indicates that the sweep is defined in a counter-clockwise direction.

RectData (variable): Either an EmfPlusRect or EmfPlusRectF object that defines the bounding box of the ellipse that is collinear with the arc. This rectangle defines the position, size, and shape of the arc. The type of object in this field is specified by the value of the **Flags** field.

See section 2.3.4 for the specification of additional drawing record types.

2.3.4.3 EmfPlusDrawBeziers Record

The EmfPlusDrawBeziers record specifies drawing a sequence of connected Bezier curves. The order for Bezier data points is the start point, control point 1, control point 2 and end point. For more information see [MSDN-DrawBeziers].



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawBeziers from the RecordType enumeration. The value MUST be 0x4019.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	C	X	X	P	X	X	X	ObjectID																							

X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the **PointData** field specifies compressed data.

If set, **PointData** specifies absolute locations in the coordinate space with 16-bit integer coordinates. If clear, **PointData** specifies absolute locations in the coordinate space with 32-bit floating-point coordinates.

Note: If the **P** flag (below) is set, this flag is undefined and MUST be ignored.

P (1 bit): This bit indicates whether the **PointData** field specifies relative or absolute locations.

If set, each element in **PointData** specifies a location in the coordinate space that is relative to the location specified by the previous element in the array. In the case of the first element in **PointData**, a previous location at coordinates (0,0) is assumed. If clear, **PointData** specifies absolute locations according to the **C** flag.

Note: If this flag is set, the **C** flag (above) is undefined and MUST be ignored.<19>

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the Bezier curves. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. At least 4 points MUST be specified.

Value	Meaning
0x00000018 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **Size** is computed as follows: `Size = (Count * 0x00000002) + 0x00000010`
0x00000020 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000004) + 0x00000010`
0x00000030 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000008) + 0x00000010`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. At least 4 points MUST be specified.

Value	Meaning
0x0000000C ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **DataSize** is computed as follows: `DataSize = (Count * 0x00000002) + 0x00000004`
0x00000014 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000004) + 0x00000004`
0x00000024 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000008) + 0x00000004`

Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** array. At least 4 points MUST be specified.

PointData (variable): An array of **Count** points that specify the starting, ending, and control points of the Bezier curves. The ending coordinate of one Bezier curve is the starting coordinate of the next. The control points are used for producing the Bezier effect.

The type of data in this array is specified by the **Flags** field, as follows:

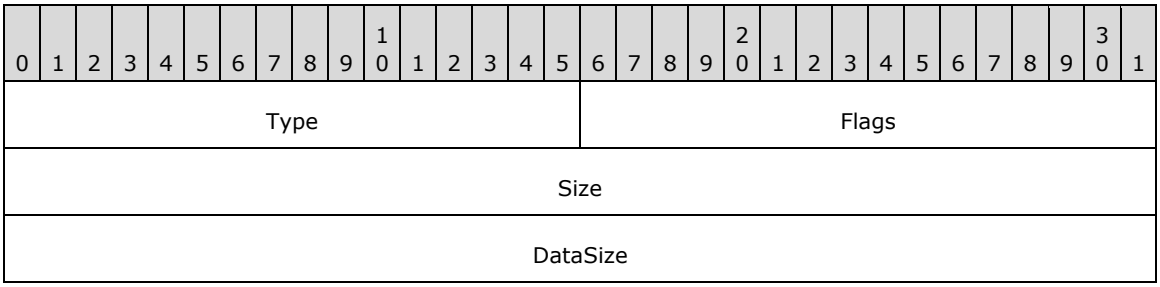
Data Type	Meaning
EmfPlusPointR object	If the **P** flag is set in the **Flags**, the points specify relative locations.
EmfPlusPointF object	If the **P** and **C** bits are clear in the **Flags** field, the points specify absolute locations.
EmfPlusPoint object	If the **P** bit is clear and the **C** bit is set in the **Flags** field, the points specify relative locations.

A Bezier curve does not pass through its control points. The control points act as magnets, pulling the curve in certain directions to influence the way the lines bend.

See section 2.3.4 for the specification of additional drawing record types.

2.3.4.4 EmfPlusDrawClosedCurve Record

The EmfPlusDrawClosedCurve record specifies drawing a closed cardinal spline.



Tension
Count
PointData (variable)
...

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawClosedCurve from the RecordType enumeration. The value MUST be 0x4017.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	C	X	X	P	X	X	X	ObjectID																							

X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the **PointData** field specifies compressed data.

If set, **PointData** specifies locations in the coordinate space with 16-bit integer coordinates. If clear, **PointData** specifies locations in the coordinate space with 32-bit floating-point coordinates.

Note: If the **P** flag (below) is set, this flag is undefined and MUST be ignored.

P (1 bit): This bit indicates whether the **PointData** field specifies relative or absolute locations.

If set, each element in **PointData** specifies a location in the coordinate space that is relative to the location specified by the previous element in the array. In the case of the first element in **PointData**, a previous location at coordinates (0,0) is assumed. If clear, **PointData** specifies absolute locations according to the **C** flag.

Note: If this flag is set, the **C** flag (above) is undefined and MUST be ignored.<20>

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the closed curve. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. At least 3 points MUST be specified.

Value	Meaning
0x0000001C ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **Size** is computed as follows: `Size = ((((Count * 0x00000002) + 0x00000014 + 0x00000003) / 4) * 4)`
0x00000020 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000004) + 0x00000014`

Value	Meaning
0x0000002C ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000008) + 0x00000014`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data in the record. At least 3 points MUST be specified.

Value	Meaning
0x00000010 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **DataSize** is computed as follows: `DataSize = ((((Count * 0x00000002) + 0x0000008 + 0x00000003) / 4) * 4)`
0x00000014 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000004) + 0x00000008`
0x00000020 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000008) + 0x00000008`

Tension (4 bytes): A 32-bit floating point number that specifies how tightly the spline bends as it passes through the points. A value of 0 specifies that the spline is a sequence of straight lines. As the value increases, the curve becomes more rounded. For more information, see [SPLINE77] and [PETZOLD].

Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** field. At least 3 points MUST be specified.

PointData (variable): An array of **Count** points that specify the endpoints of the lines that define the spline. In a closed cardinal spline, the curve continues through the last point in the **PointData** array and connects with the first point in the array.

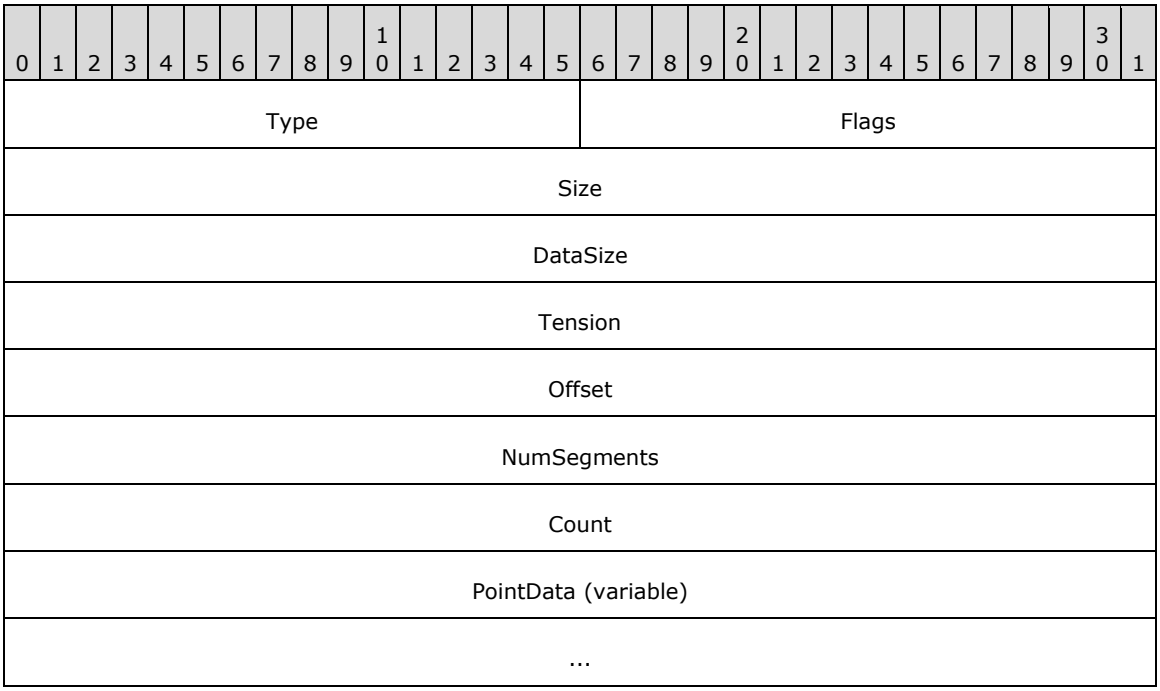
The type of data in this array is specified by the **Flags** field, as follows:

Data Type	Meaning
EmfPlusPointR object	If the **P** bit is set in the **Flags** field, the points specify relative locations.
EmfPlusPoint object	If the **P** bit is clear and the **C** bit is set in the **Flags** field, the points specify absolute locations with 16-bit coordinates.
EmfPlusPointF object	If the **P** and **C** bits are clear in the **Flags** field, the points specify absolute locations with 32-bit coordinates.

See section 2.3.4 for the specification of additional drawing record types.

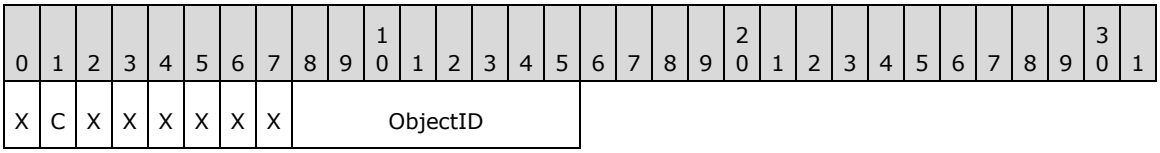
2.3.4.5 EmfPlusDrawCurve Record

The EmfPlusDrawCurve record specifies drawing a cardinal spline.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawCurve from the RecordType enumeration. The value MUST be 0x4018.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the data in the **PointData** field is compressed.

If set, **PointData** contains an array of EmfPlusPoint objects. If clear, **PointData** contains an array of EmfPlusPointF objects.

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the curve. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data.

At least 2 **PointData** elements MUST be specified in this record.

Value	Meaning
0x00000024 ≤ *value*	If the **C** bit is set in the **Flags** field, **Count** points with values of 16-bit signed integers are defined in the **PointData** field. In this case, **Size** MUST be computed as

Value	Meaning
	follows: `Size = (Count * 0x00000004) + 0x0000001C`
0x0000002C ≤ *value*	If the **C** bit is clear in the **Flags** field, **Count** points with values of 32-bit floating-point numbers are defined in the **PointData** field. In this case, **Size** MUST be computed as follows: `Size = (Count * 0x00000008) + 0x0000001C`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows.

At least 2 **PointData** elements MUST be specified in this record.

Value	Meaning
0x00000018 ≤ *value*	If the **C** bit is set in the **Flags** field, **Count** points with values of 16-bit signed integers are defined in the **PointData** field. In this case, **DataSize** MUST be computed as follows: `DataSize = (Count * 0x00000004) + 0x00000010`
0x00000020 ≤ *value*	If the **C** bit is clear in the **Flags** field, **Count** points with values of 32-bit floating-point numbers are defined in the **PointData** field. In this case, **DataSize** MUST be computed as follows: `DataSize = (Count * 0x00000008) +  0x00000010`

Tension (4 bytes): A 32-bit floating-point value that specifies how tightly the spline bends as it passes through the points. A value of 0 specifies that the spline is a sequence of straight lines. As the value increases, the curve becomes more rounded. For more information, see [SPLINE77] and [PETZOLD].

Offset (4 bytes): A 32-bit unsigned integer that specifies the element in the **PointData** array that defines the starting point of the spline.

NumSegments (4 bytes): A 32-bit unsigned integer that specifies the number of line segments making up the spline.

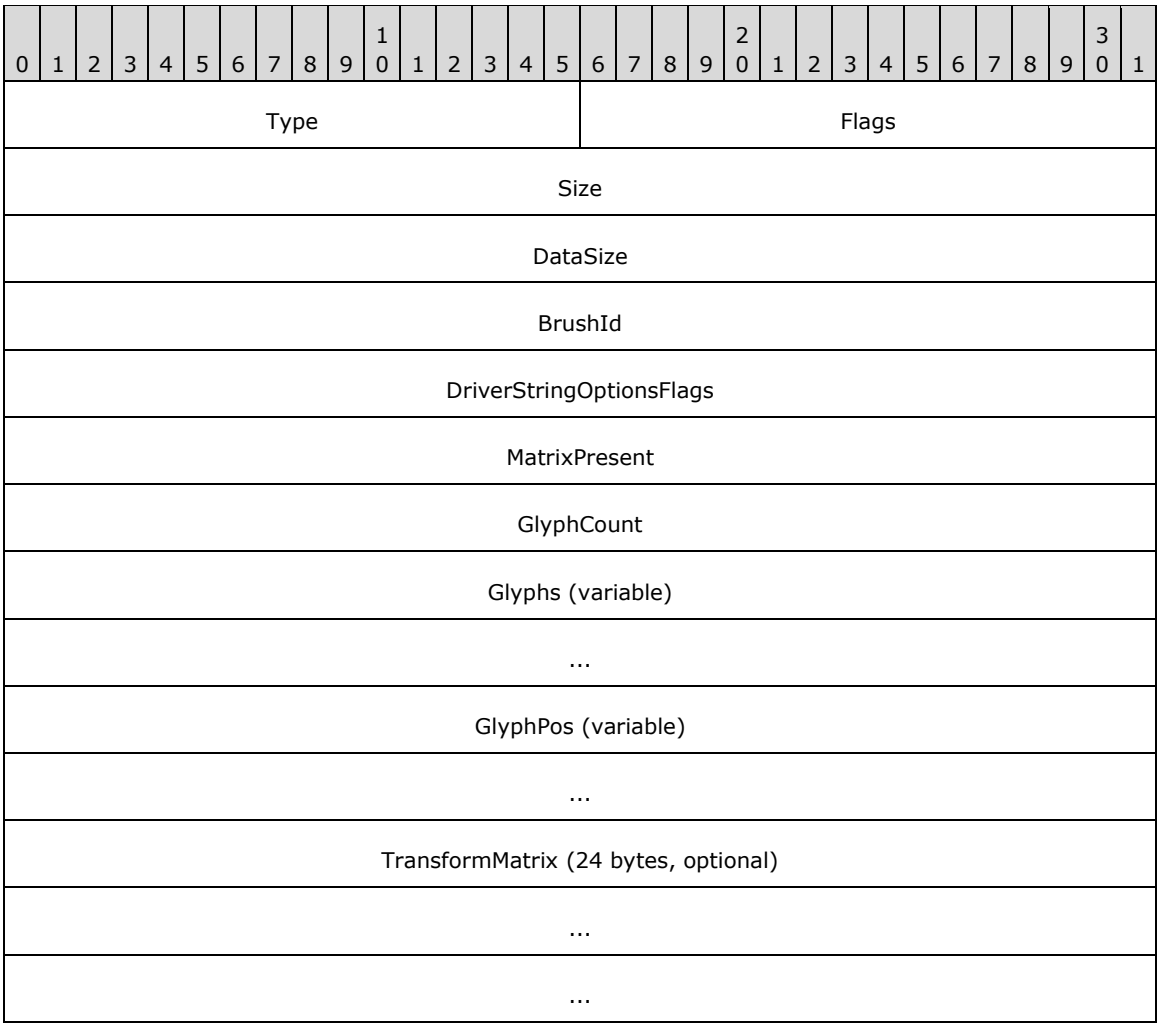
Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** array. The minimum number of points for drawing a curve is 2—the starting and ending points.

PointData (variable): An array of either 32-bit signed integers or 32-bit floating-point numbers of **Count** length that defines coordinate values of the endpoints of the lines to be stroked.

See section 2.3.4 for the specification of additional drawing record types.

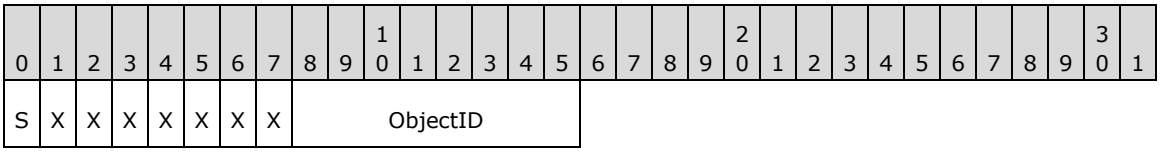
2.3.4.6 EmfPlusDrawDriverString Record

The EmfPlusDrawDriverString record specifies text output with character positions.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawDriverString from the RecordType enumeration. The value MUST be 0x4036.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



S (1 bit): This bit indicates the type of data in the **BrushId** field.

If set, **BrushId** specifies the color value in an EmfPlusARGB object. If clear, **BrushId** contains the EMF+ Object Table index of an EmfPlusBrush object.

X (1 bit): Reserved and MUST be ignored.

ObjectID (1 byte): The EMF+ Object Table index of an EmfPlusFont object to render the text. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data.

Value	Meaning
0x0000001C ≤ *value*	When glyphs are provided, but no transform matrix is specified in the **TransformMatrix** field, the size of the record is computed as follows: `Size = (GlyphCount * 0x0000000A) + 0x0000001C`
0x00000034 ≤ *value*	When glyphs are provided, and a transform matrix is specified in the **TransformMatrix** field, the size of the record is computed as follows: `Size = (GlyphCount * 0x0000000A) + 0x00000034`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows.

Value	Meaning
0x00000010 ≤ *value*	When glyphs are provided, but no transform matrix is specified in the **TransformMatrix** field, the size of the data is computed as follows: `DataSize = (GlyphCount * 0x0000000A) + 0x00000010`
0x00000028 ≤ *value*	When glyphs are provided, and a transform matrix is specified in the **TransformMatrix** field, the size of the data is computed as follows: `DataSize = (GlyphCount * 0x0000000A) + 0x00000028`

BrushId (4 bytes): A 32-bit unsigned integer that specifies either the foreground color of the text or a graphics brush, depending on the value of the **S** flag in the **Flags**.

DriverStringOptionsFlags (4 bytes): A 32-bit unsigned integer that specifies the spacing, orientation, and quality of rendering for the string. This value MUST be composed of DriverStringOptions flags.

MatrixPresent (4 bytes): A 32-bit unsigned integer that specifies whether a transform matrix is present in the **TransformMatrix** field.

Value	Meaning
0x00000000	The transform matrix is not present in the record.
0x00000001	The transform matrix is present in the record.

GlyphCount (4 bytes): A 32-bit unsigned integer that specifies number of glyphs in the string.

Glyphs (variable): An array of 16-bit values that define the text string to draw.

If the **DriverStringOptionsCmapLookup** flag in the **DriverStringOptionsFlags** field is set, each value in this array specifies a Unicode character. Otherwise, each value specifies an index to a character glyph in the EmfPlusFont object specified by the **ObjectId** value in **Flags** field.

GlyphPos (variable): An array of EmfPlusPointF objects that specify the output position of each character glyph. There MUST be **GlyphCount** elements, which have a one-to-one correspondence with the elements in the **Glyphs** array.

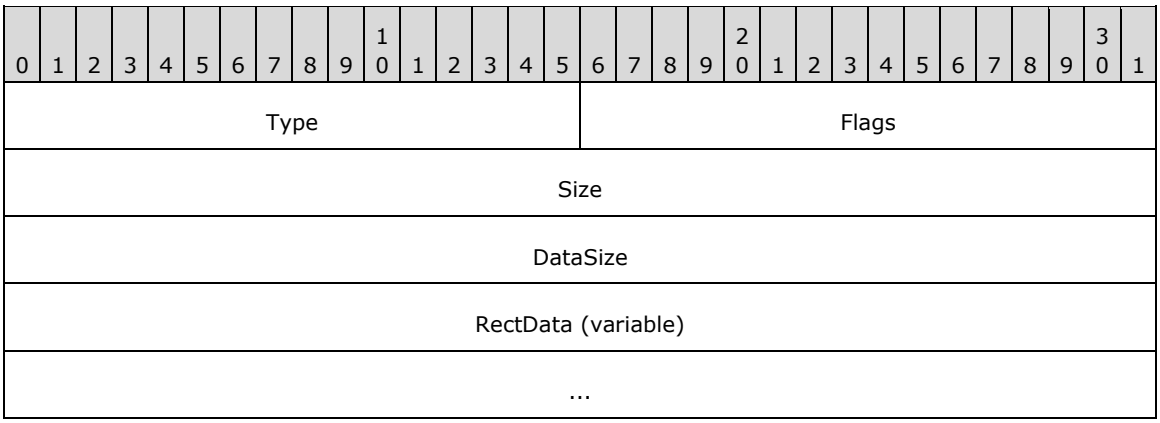
Glyph positions are calculated from the position of the first glyph if the DriverStringOptionsRealizedAdvance flag in DriverStringOptions flags is set. In this case, **GlyphPos** specifies the position of the first glyph only.

TransformMatrix (24 bytes): An optional EmfPlusTransformMatrix object that specifies the transformation to apply to each value in the text array. The presence of this data is determined from the **MatrixPresent** field.

See section 2.3.4 for the specification of additional drawing record types.

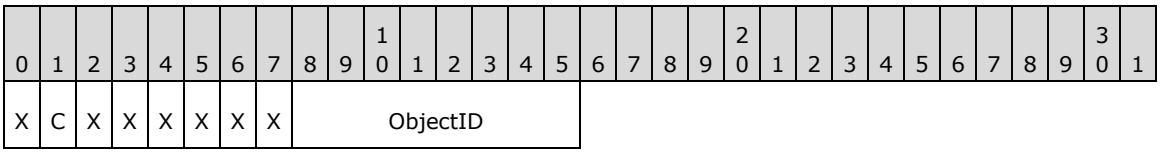
2.3.4.7 EmfPlusDrawEllipse Record

The EmfPlusDrawEllipse record specifies drawing an ellipse.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawEllipse from the RecordType enumeration. The value MUST be 0x400F.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the ellipse. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000014	If the **C** bit is set in the **Flags** field.
0x0000001C	If the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following:

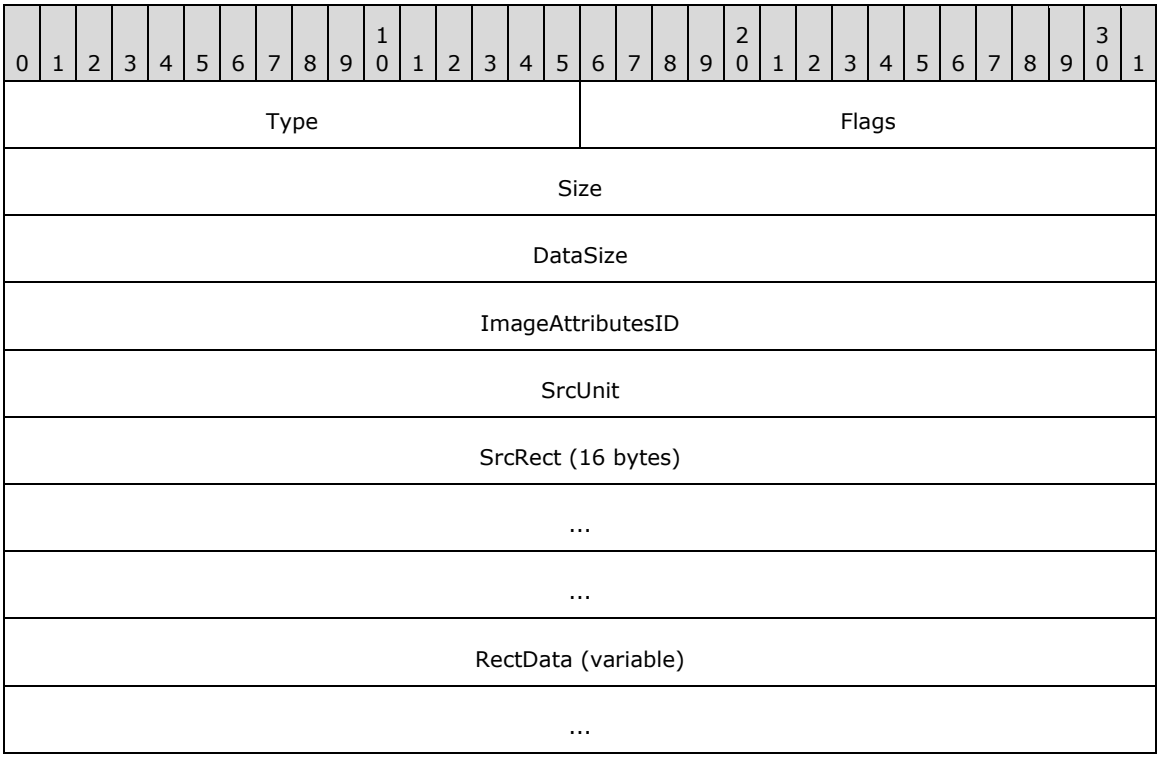
Value	Meaning
0x00000008	If the **C** bit is set in the **Flags** field.
0x00000010	If the **C** bit is clear in the **Flags** field.

RectData (variable): Either an EmfPlusRect or EmfPlusRectF object that defines the bounding box of the ellipse.

See section 2.3.4 for the specification of additional drawing record types.

2.3.4.8 EmfPlusDrawImage Record

The EmfPlusDrawImage record specifies drawing a scaled image.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawImage from the RecordType enumeration. The value MUST be 0x401A.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	C	X	X	X	X	X	X	ObjectID																							

X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

ObjectID (1 byte): The index of an EmfPlusImage object in the EMF+ Object Table, which specifies the image to render. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be one of the following:

Value	Meaning
0x0000002C	If the **C** bit is set in the **Flags** field.
0x00000034	If the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000020	If the **C** bit is set in the **Flags** field.
0x00000028	If the **C** bit is clear in the **Flags** field.

ImageAttributesID (4 bytes): A 32-bit unsigned integer that specifies the index of an optional EmfPlusImageAttributes object in the EMF+ Object Table.

SrcUnit (4 bytes): A 32-bit signed integer that specifies the units of the **SrcRect** field. It MUST be the **UnitTypePixel** member of the UnitType enumeration.

SrcRect (16 bytes): An EmfPlusRectF object that specifies a portion of the image to be rendered. The portion of the image specified by this rectangle is scaled to fit the destination rectangle specified by the **RectData** field.

RectData (variable): Either an EmfPlusRect or EmfPlusRectF object that defines the bounding box of the image. The portion of the image specified by the **SrcRect** field is scaled to fit this rectangle.

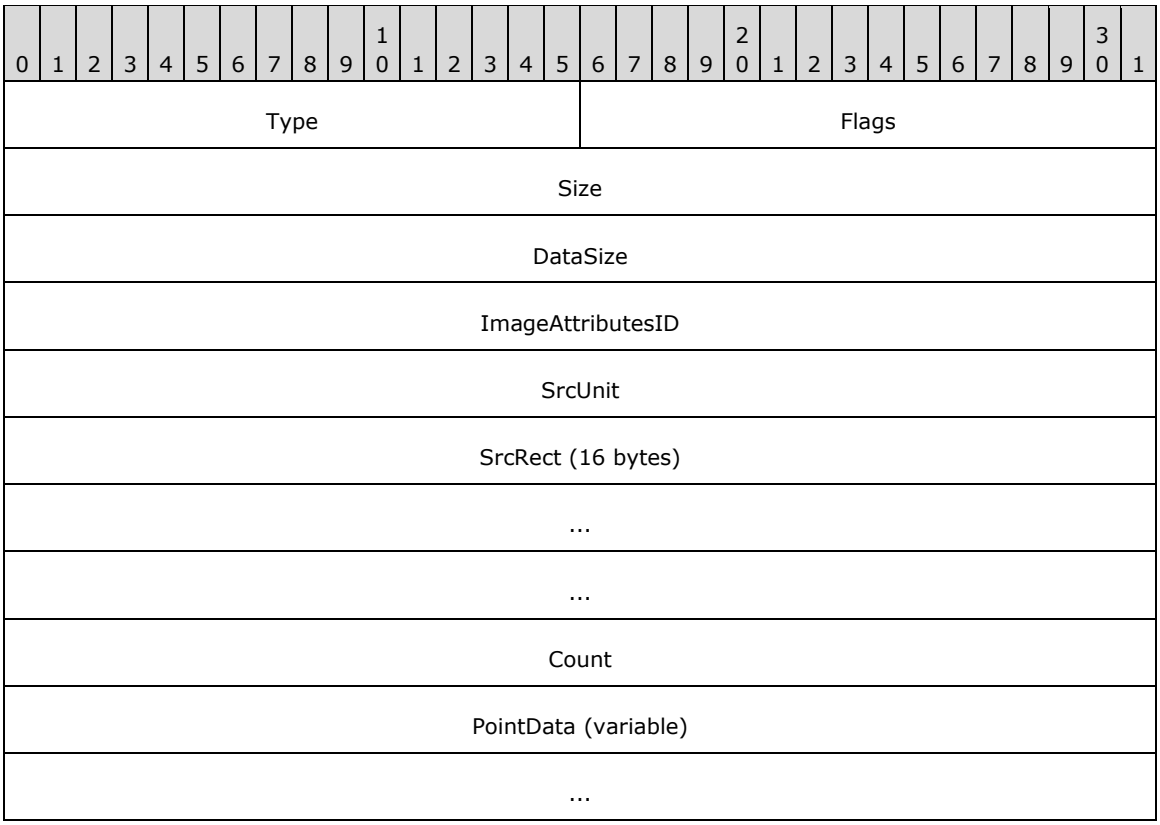
An EmfPlusImage object can specify either a bitmap or a metafile.

Colors in an image can be manipulated during rendering. They can be corrected, darkened, lightened, and removed.

See section 2.3.4 for the specification of additional drawing record types.

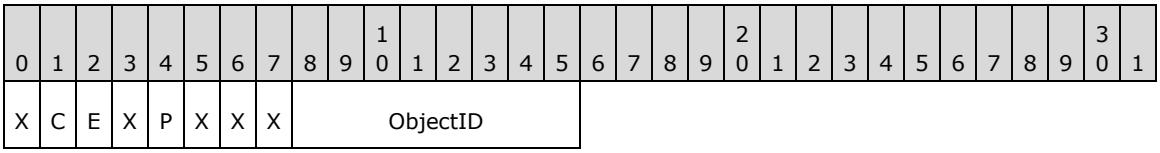
2.3.4.9 EmfPlusDrawImagePoints Record

The EmfPlusDrawImagePoints record specifies drawing a scaled image inside a parallelogram.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawImagePoints from the RecordType enumeration. The value MUST be 0x401B.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the **PointData** field specifies compressed data.

If set, **PointData** specifies absolute locations in the coordinate space with 16-bit integer coordinates. If clear, **PointData** specifies absolute locations in the coordinate space with 32-bit floating-point coordinates.

Note: If the **P** flag (below) is set, this flag is undefined and MUST be ignored.

E (1 bit): This bit indicates that the rendering of the image includes applying an effect.

If set, an object of the **Effect** class MUST have been specified in an earlier EmfPlusSerializableObject record.

P (1 bit): This bit indicates whether the **PointData** field specifies relative or absolute locations.

If set, each element in **PointData** specifies a location in the coordinate space that is relative to the location specified by the previous element in the array. In the case of the first element in

PointData, a previous location at coordinates (0,0) is assumed. If clear, **PointData** specifies absolute locations according to the **C** flag.

Note: If this flag is set, the **C** flag (above) is undefined and MUST be ignored.<21>

ObjectID (1 byte): The index of an EmfPlusImage object in the EMF+ Object Table, which specifies the image to render. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. For this record type, the value MUST be one of the following.

Value	Meaning
0x00000030	If the **P** bit is set in the **Flags** field.
0x00000034	If the **P** bit is clear and the **C** bit is set in the **Flags** field.
0x00000040	If the **P** bit is clear and the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following.

Value	Meaning
0x00000024	If the **P** bit is set in the **Flags** field.
0x00000028	If the **P** bit is clear and the **C** bit is set in the **Flags** field.
0x00000034	If the **P** bit is clear and the **C** bit is clear in the **Flags** field.

ImageAttributesID (4 bytes): A 32-bit unsigned integer that contains the index of the optional EmfPlusImageAttributes object in the EMF+ Object Table.

SrcUnit (4 bytes): A 32-bit signed integer that defines the units of the **SrcRect** field. It MUST be the **UnitPixel** value of the UnitType enumeration.

SrcRect (16 bytes): An EmfPlusRectF object that defines a portion of the image to be rendered.

Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** array. Exactly 3 points MUST be specified.

PointData (variable): An array of **Count** points that specify three points of a parallelogram. The three points represent the upper-left, upper-right, and lower-left corners of the parallelogram. The fourth point of the parallelogram is extrapolated from the first three. The portion of the image specified by the **SrcRect** field SHOULD have scaling and shearing transforms applied if necessary to fit inside the parallelogram.

The type of data in this array is specified by the **Flags** field, as follows.

Data Type	Meaning
EmfPlusPointR object	If the **P** flag is set in the **Flags**, the points specify relative locations.
EmfPlusPoint object	If the **P** bit is clear and the **C** bit is set in the **Flags** field, the points specify absolute locations with integer values.
EmfPlusPointF object	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, the points specify absolute locations with floating-point values.

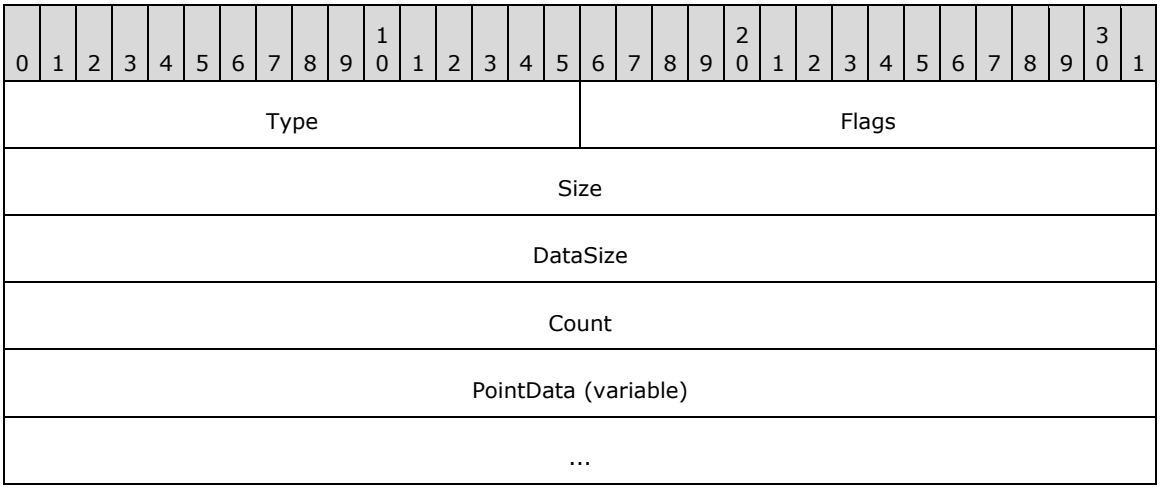
An EmfPlusImage can specify either a bitmap or metafile.

Colors in an image can be manipulated during rendering. They can be corrected, darkened, lightened, and removed.

See section 2.3.4 for the specification of additional drawing record types.

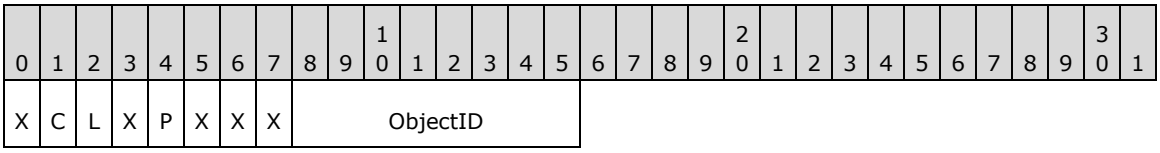
2.3.4.10 EmfPlusDrawLines Record

The EmfPlusDrawlLines record specifies drawing a series of connected lines.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawlLines from the RecordType enumeration. The value MUST be 0x400D.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the **PointData** field specifies compressed data.

If set, **PointData** specifies absolute locations in the coordinate space with 16-bit integer coordinates. If clear, **PointData** specifies absolute locations in the coordinate space with 32-bit floating-point coordinates.

Note: If the **P** flag (below) is set, this flag is undefined and MUST be ignored.

L (1 bit): This bit indicates whether to draw an extra line between the last point and the first point, to close the shape.

P (1 bit): This bit indicates whether the **PointData** field specifies relative or absolute locations.

If set, each element in **PointData** specifies a location in the coordinate space that is relative to the location specified by the previous element in the array. In the case of the first element in **PointData**, a previous location at coordinates (0,0) is assumed. If clear, **PointData** specifies absolute locations according to the **C** flag.

Note: If this flag is set, the **C** flag (above) is undefined and MUST be ignored.<22>

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the lines. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. For this record type, the value MUST be one of the following.

Value	Meaning
0x00000014 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **Size** is computed as follows: `Size = (Count * 0x00000002) + 0x00000010`
0x00000018 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000004) + 0x00000010`
0x00000020 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000008) + 0x00000010`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following.

Value	Meaning
0x00000008 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **DataSize** is computed as follows: `DataSize = (Count * 0x00000002) + 0x00000004`
0x0000000C ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000004) + 0x00000004`
0x00000014 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000008) + 0x00000004`

Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** array. At least 2 points MUST be specified.

PointData (variable): An array of **Count** points that specify the starting and ending points of the lines to be drawn.

The type of data in this array is specified by the **Flags** field, as follows.

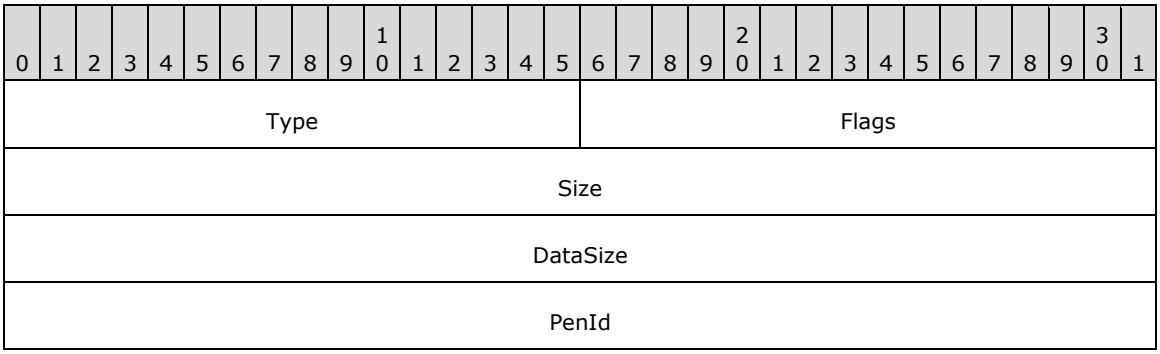
Data Type	Meaning
EmfPlusPointR object	If the **P** flag is set in the **Flags**, the points specify relative locations.

Data Type	Meaning
EmfPlusPoint object	If the **P** bit is clear and the **C** bit is set in the **Flags** field, the points specify absolute locations with integer values.
EmfPlusPointF object	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, the points specify absolute locations with floating-point values.

See section 2.3.4 for the specification of additional drawing record types.

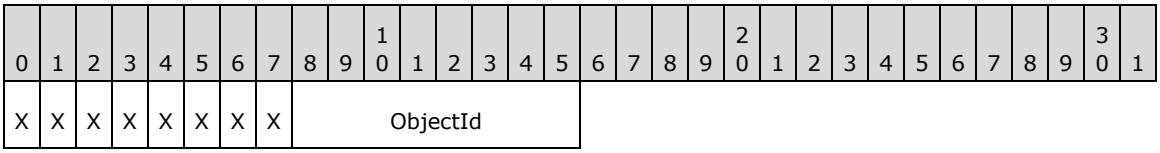
2.3.4.11 EmfPlusDrawPath Record

The EmfPlusDrawPath record specifies drawing a graphics path.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawPath from the RecordType enumeration. The value MUST be 0x4015.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

ObjectId (1 byte): The index of the EmfPlusPath object to draw, in the EMF+ Object Table. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

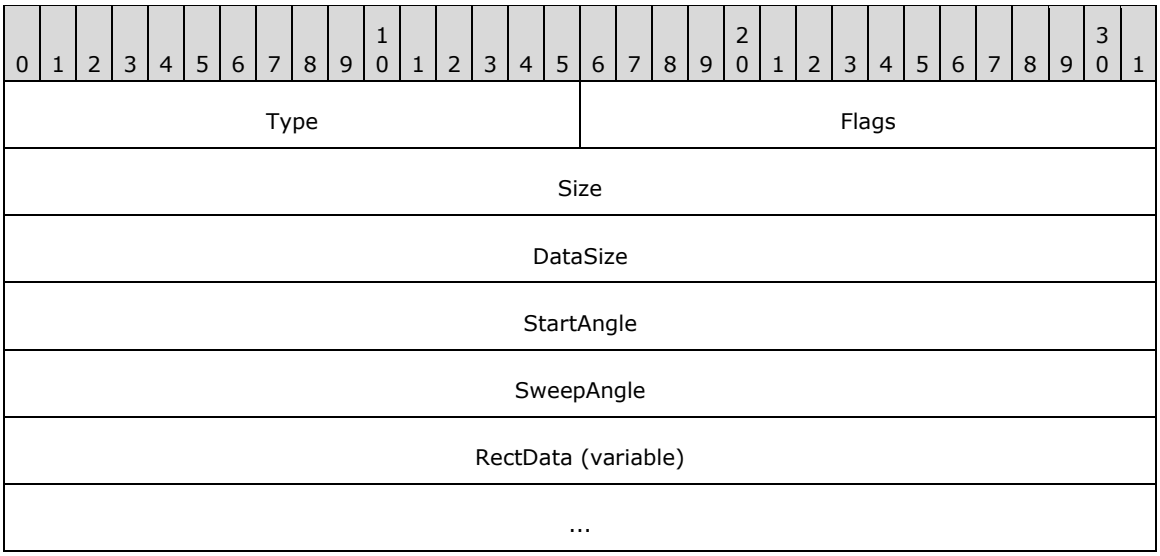
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value is 0x00000004.

PenId (4 bytes): A 32-bit unsigned integer that specifies an index in the EMF+ Object Table for an EmfPlusPen object to use for drawing the EmfPlusPath. The value MUST be zero to 63, inclusive.

See section 2.3.4 for the specification of additional drawing record types.

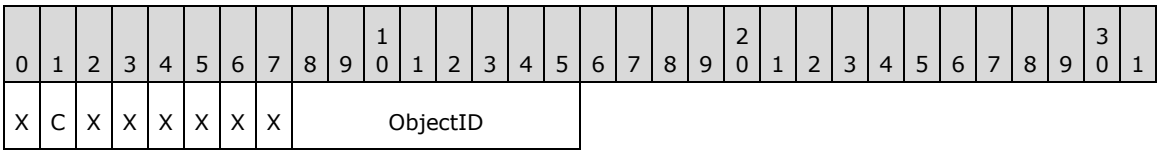
2.3.4.12 EmfPlusDrawPie Record

The EmfPlusDrawPie record specifies drawing a section of the interior of an ellipse.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawPie from the RecordType enumeration. The value MUST be 0x4011.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the pie. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be one of the following:

Value	Meaning
0x0000001C	If the **C** bit is set in the **Flags** field.
0x00000024	If the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000010	If the **C** bit is set in the **Flags** field.

Value	Meaning
0x00000018	If the **C** bit is clear in the **Flags** field.

StartAngle (4 bytes): A 32-bit, non-negative floating-point value that specifies the angle between the x-axis and the starting point of the pie wedge. Any value is acceptable, but it MUST be interpreted modulo 360, with the result that is used being in the range 0.0 inclusive to 360.0 exclusive.

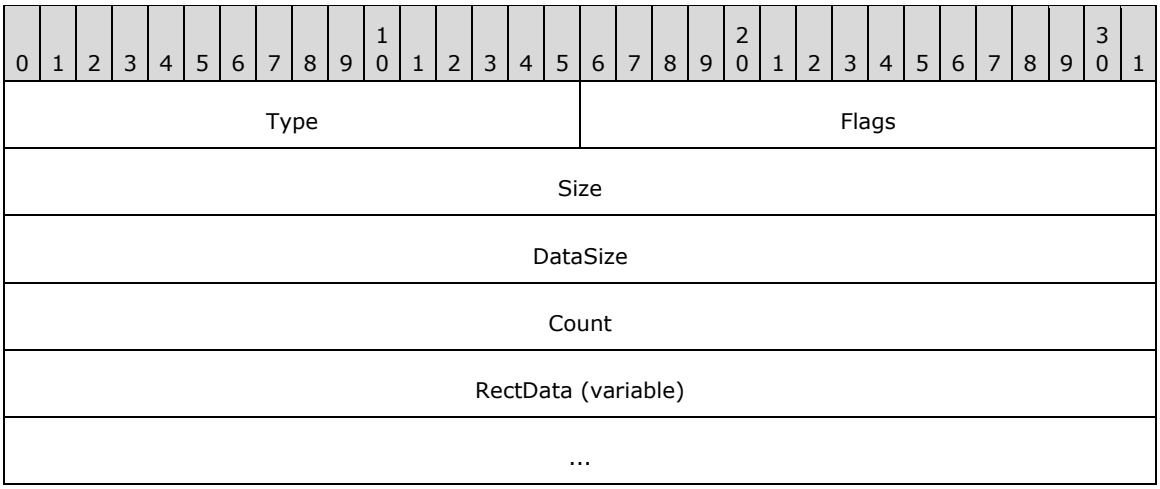
SweepAngle (4 bytes): A 32-bit floating-point value that specifies the extent of the arc that defines the pie wedge to draw, as an angle in degrees measured from the starting point defined by the **StartAngle** value. Any value is acceptable, but it MUST be clamped to -360.0 to 360.0 inclusive. A positive value indicates that the sweep is defined in a clockwise direction, and a negative value indicates that the sweep is defined in a counter-clockwise direction.

RectData (variable): Either an EmfPlusRect or EmfPlusRectF object that defines the bounding box of the ellipse that contains the pie wedge. This rectangle defines the position, size, and shape of the pie. The type of object in this field is specified by the value of the **Flags** field.

See section 2.3.4 for the specification of additional drawing record types.

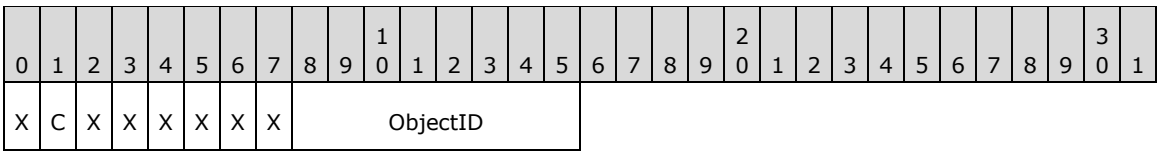
2.3.4.13 EmfPlusDrawRects Record

The EmfPlusDrawRects record specifies drawing a series of rectangles.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawRects from the RecordType enumeration. The value MUST be 0x400B.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

ObjectID (1 byte): The index of an EmfPlusPen object in the EMF+ Object Table to draw the rectangles. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data.

At least 1 **RectData** array element MUST be specified in this record.

Value	Meaning
0x00000018 ≤ *value*	If the **C** bit is set in the **Flags** field, **Size** MUST be computed as follows: `Size = (Count * 0x00000008) + 0x00000010`
0x00000020 ≤ *value*	If the **C** bit is clear in the **Flags** field, **Size** MUST be computed as follows: `Size = (Count * 0x00000010) + 0x00000010`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows.

At least 1 **RectData** array element MUST be specified in this record.

Value	Meaning
0x0000000C ≤ *value*	If the **C** bit is set in the **Flags** field, **DataSize** MUST be computed as follows: `DataSize = (Count * 0x00000008) + 0x00000004`
0x00000014 ≤ *value*	If the **C** bit is clear in the **Flags** field, **DataSize** MUST be computed as follows: `DataSize = (Count * 0x00000010) + 0x00000004`

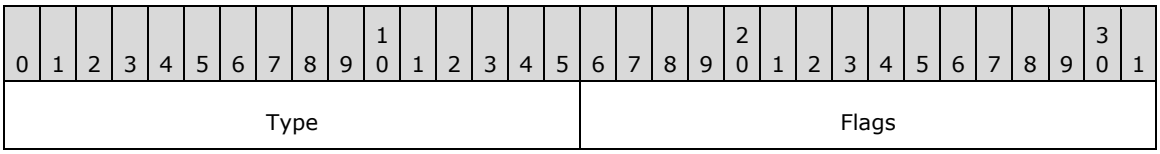
Count (4 bytes): A 32-bit unsigned integer that specifies the number of rectangles in the **RectData** member.

RectData (variable): An array of either an EmfPlusRect or EmfPlusRectF objects of **Count** length that defines the rectangle data.

See section 2.3.4 for the specification of additional drawing record types.

2.3.4.14 EmfPlusDrawString Record

The EmfPlusDrawString record specifies text output with string formatting.



Size
DataSize
BrushId
FormatID
Length
LayoutRect (16 bytes)
...
...
StringData (variable)
...
AlignmentPadding (variable)
...

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusDrawString from the RecordType enumeration. The value MUST be 0x401C.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S	X	X	X	X	X	X	X	ObjectID																							

S (1 bit): This bit indicates the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

X (1 bit): Reserved and MUST be ignored.

ObjectID (1 byte): The index of an EmfPlusFont object in the EMF+ Object Table to render the text. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header, record-specific data, and any extra alignment padding.

Value	Meaning
0x0000002A ≤ *value*	The size of the record is computed as follows: `Size = (Length * 0x00000002) + 0x00000028 (+ AlignmentPaddingSize`

Value	Meaning
	where AlignmentPaddingSize is the number of bytes in AlignmentPadding)

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data and any extra alignment padding that follows.

Value	Meaning
0x0000001E ≤ *value*	The size of the data is computed as follows: DataSize = (Length * 0x00000002) + 0x0000001C (+ AlignmentPaddingSize where AlignmentPaddingSize is the number of bytes in AlignmentPadding)

BrushId (4 bytes): A 32-bit unsigned integer that specifies the brush, the content of which is determined by the **S** bit in the **Flags** field. This definition is used to paint the foreground text color; that is, just the glyphs themselves.

FormatID (4 bytes): A 32-bit unsigned integer that specifies the index of an optional EmfPlusStringFormat object in the EMF+ Object Table. This object specifies text layout information and display manipulations to be applied to a string.

Length (4 bytes): 32-bit unsigned integer that specifies the number of characters in the string.

LayoutRect (16 bytes): An EmfPlusRectF object that defines the bounding area of the destination that will receive the string.

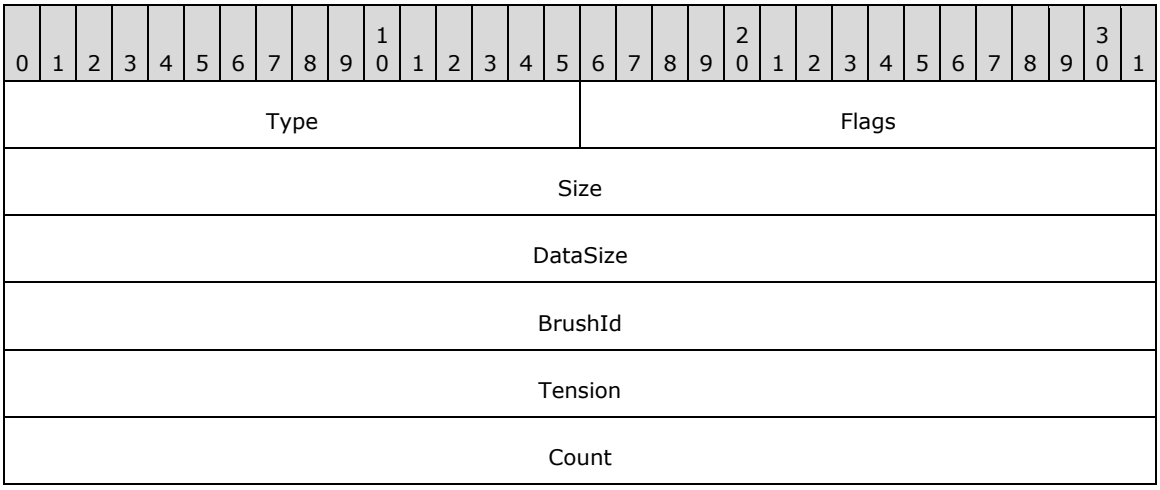
StringData (variable): An array of 16-bit Unicode characters that specifies the string to be drawn.

AlignmentPadding (variable): An optional array of up to 3 bytes that pads the record-specific data so that **DataSize** is a multiple of 4 bytes. This field MUST be ignored.

See section 2.3.4 for the specification of additional drawing record types.

2.3.4.15 EmfPlusFillClosedCurve Record

The EmfPlusFillClosedCurve record specifies filling the interior of a closed cardinal spline.



PointData (variable)
...

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillClosedCurve from the RecordType enumeration. The value MUST be 0x4016.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of 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
S	C	W	X	P	X	X	X	X	X	X	X	X	X	X	X																

S (1 bit): This bit indicates the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

C (1 bit): This bit indicates whether the **PointData** field specifies compressed data.

If set, **PointData** specifies absolute locations in the coordinate space with 16-bit integer coordinates. If clear, **PointData** specifies absolute locations in the coordinate space with 32-bit floating-point coordinates.

Note: If the **P** flag (below) is set, this flag is undefined and MUST be ignored.

W (1 bit): This bit indicates how to perform the fill operation.

If set, the fill is a "winding" fill. If clear, the fill is an "alternate" fill.

X (1 bit): Reserved and MUST be ignored.

P (1 bit): This bit indicates whether the **PointData** field specifies relative or absolute locations.

If set, each element in **PointData** specifies a location in the coordinate space that is relative to the location specified by the previous element in the array. In the case of the first element in **PointData**, a previous location at coordinates (0,0) is assumed. If clear, **PointData** specifies absolute locations according to the **C** flag.

Note: If this flag is set, the **C** flag (above) is undefined and MUST be ignored.<23>

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. At least 3 points MUST be specified.

Value	Meaning
0x00000020 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **Size** is computed as follows: `Size = ((((Count * 0x00000002) + 0x00000018 + 0x00000003) / 4) * 4)`
0x00000024 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000004) + 0x00000018`

Value	Meaning
0x00000030 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000008) + 0x00000018`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. At least 3 points MUST be specified.

Value	Meaning
0x00000014 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **DataSize** is computed as follows: `DataSize = ((((Count * 0x00000002) + 0x0000000C + 0x00000003) / 4) * 4)`
0x00000018 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000004) + 0x0000000C`
0x00000024 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000008) + 0x0000000C`

BrushId (4 bytes): A 32-bit unsigned integer that specifies the EmfPlusBrush, the content of which is determined by the **S** bit in the **Flags** field. This brush is used to fill the interior of the closed cardinal spline.

Tension (4 bytes): A 32-bit floating point value that specifies how tightly the spline bends as it passes through the points. A value of 0.0 specifies that the spline is a sequence of straight lines. As the value increases, the curve becomes more rounded. For more information, see [SPLINE77] and [PETZOLD].

Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** field. At least 3 points MUST be specified.

PointData (variable): An array of **Count** points that specify the endpoints of the lines that define the spline. In a closed cardinal spline, the curve continues through the last point in the **PointData** array and connects with the first point in the array.

The type of data in this array is specified by the **Flags** field, as follows:

Data Type	Meaning
EmfPlusPointR object	If the **P** flag is set in the **Flags**, the points specify relative locations.
EmfPlusPoint object	If the **P** bit is clear and the **C** bit is set in the **Flags** field, the points specify absolute locations with integer values.
EmfPlusPointF object	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, the points specify absolute locations with floating-point values.

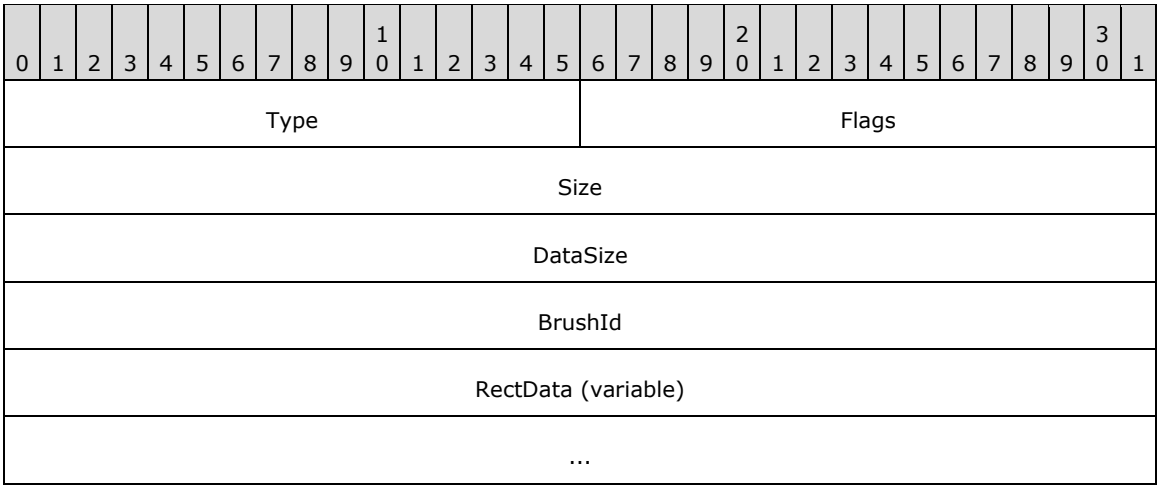
A "winding" fill operation fills areas according to the "even-odd parity" rule. According to this rule, a test point can be determined to be inside or outside a closed curve as follows: Draw a line from the test point to a point that is distant from the curve. If that line crosses the curve an odd number of times, the test point is inside the curve; otherwise, the test point is outside the curve.

An "alternate" fill operation fills areas according to the "non-zero" rule. According to this rule, a test point can be determined to be inside or outside a closed curve as follows: Draw a line from a test point to a point that is distant from the curve. Count the number of times the curve crosses the test line from left to right, and count the number of times the curve crosses the test line from right to left. If those two numbers are the same, the test point is outside the curve; otherwise, the test point is inside the curve.

See section 2.3.4 for the specification of additional drawing record types.

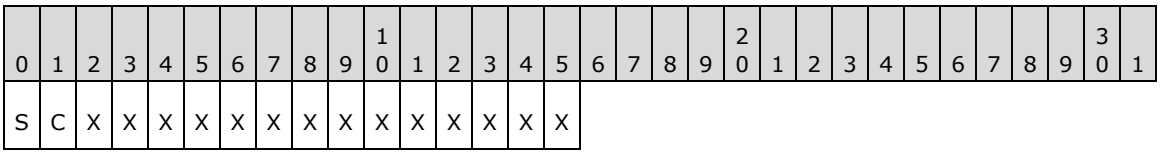
2.3.4.16 EmfPlusFillEllipse Record

The EmfPlusFillEllipse record specifies filling the interior of an ellipse.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillEllipse from the RecordType enumeration. The value MUST be 0x400E.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



S (1 bit): This bit specifies the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

X (1 bit): Reserved and MUST be ignored.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000018	If the **C** bit is set in the **Flags** field.
0x00000020	If the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following:

Value	Meaning
0x0000000C	If the **C** bit is set in the **Flags** field.
0x00000014	If the **C** bit is clear in the **Flags** field.

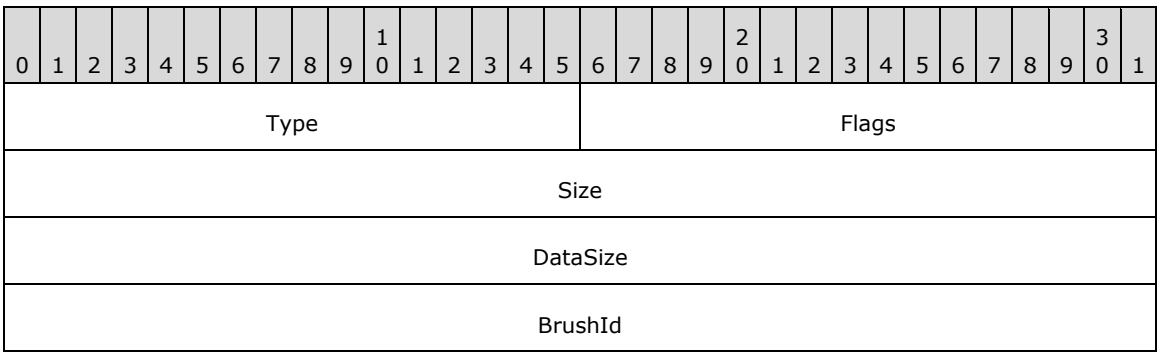
BrushId (4 bytes): A 32-bit unsigned integer that specifies the brush, the content of which is determined by the **S** bit in the **Flags** field. This definition is used to fill the interior of the ellipse.

RectData (variable): Either an EmfPlusRect or EmfPlusRectF object that defines the bounding box of the ellipse.

See section 2.3.4 for the specification of additional drawing record types.

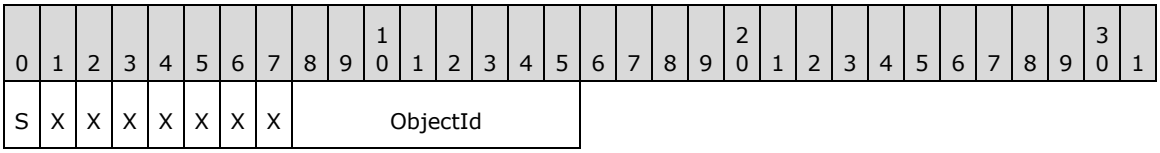
2.3.4.17 EmfPlusFillPath Record

The EmfPlusFillPath record specifies filling the interior of a graphics path.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillPath from the RecordType enumeration. The value MUST be 0x4014.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



S (1 bit): This bit indicates the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

X (1 bit): Reserved and MUST be ignored.

ObjectId (1 byte): The index of the EmfPlusPath object to fill, in the EMF+ Object Table. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

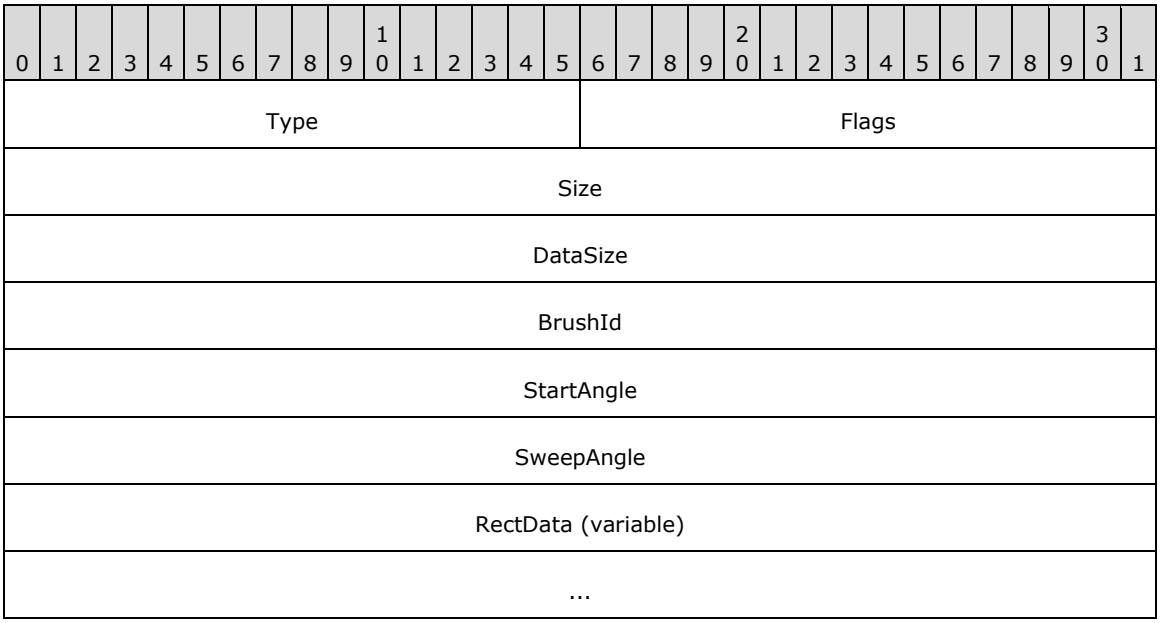
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data of record-specific data that follows. For this record type, the value MUST be 0x00000004.

BrushId (4 bytes): A 32-bit unsigned integer that defines the brush, the content of which is determined by the **S** bit in the **Flags** field.

See section 2.3.4 for the specification of additional drawing record types.

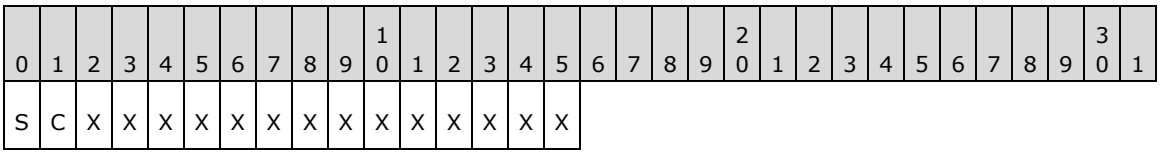
2.3.4.18 EmfPlusFillPie Record

The EmfPlusFillPie record specifies filling a section of the interior of an ellipse.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillPie from the RecordType enumeration. The value MUST be 0x4010.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



S (1 bit): This bit indicates the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

X (1 bit): Reserved and MUST be ignored.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000020	If the **C** bit is set in the **Flags** field.
0x00000028	If the **C** bit is clear in the **Flags** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be one of the following:

Value	Meaning
0x00000014	If the **C** bit is set in the **Flags** field.
0x0000001C	If the **C** bit is clear in the **Flags** field.

BrushId (4 bytes): A 32-bit unsigned integer that defines the brush, the content of which is determined by the **S** bit in the **Flags** field.

StartAngle (4 bytes): A 32-bit, non-negative floating-point value that specifies the angle between the x-axis and the starting point of the pie wedge. Any value is acceptable, but it MUST be interpreted modulo 360, with the result that is used being in the range 0.0 inclusive to 360.0 exclusive.

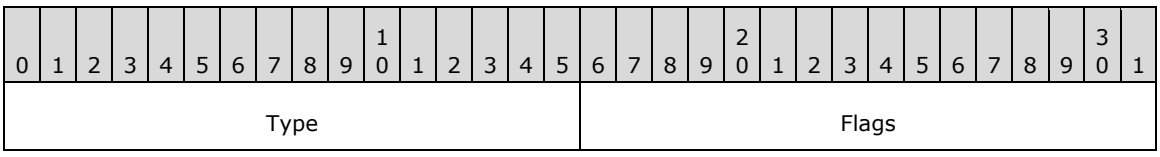
SweepAngle (4 bytes): A 32-bit floating-point value that specifies the extent of the arc that defines the pie wedge to fill, as an angle in degrees measured from the starting point defined by the **StartAngle** value. Any value is acceptable, but it MUST be clamped to -360.0 to 360.0 inclusive. A positive value indicates that the sweep is defined in a clockwise direction, and a negative value indicates that the sweep is defined in a counter-clockwise direction.

RectData (variable): Either an EmfPlusRect or EmfPlusRectF object that defines the bounding box of the ellipse that contains the pie wedge. This rectangle defines the position, size, and shape of the pie. The type of object in this field is specified by the value of the **Flags** field.

See section 2.3.4 for the specification of additional drawing record types.

2.3.4.19 EmfPlusFillPolygon Record

The EmfPlusFillPolygon record specifies filling the interior of a polygon.



Size
DataSize
BrushId
Count
PointData (variable)
...

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillPolygon from the RecordType enumeration. The value MUST be 0x400C.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S	C	X	X	P	X	X	X	X	X	X	X	X	X	X	X																

S (1 bit): This bit indicates the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

C (1 bit): This bit indicates whether the **PointData** field specifies compressed data.

If set, **PointData** specifies absolute locations in the coordinate space with 16-bit integer coordinates. If clear, **PointData** specifies absolute locations in the coordinate space with 32-bit floating-point coordinates.

Note: If the **P** flag (below) is set, this flag is undefined and MUST be ignored.

X (1 bit): Reserved and MUST be ignored.

P (1 bit): This bit indicates whether the **PointData** field specifies relative or absolute locations.

If set, each element in **PointData** specifies a location in the coordinate space that is relative to the location specified by the previous element in the array. In the case of the first element in **PointData**, a previous location at coordinates (0,0) is assumed. If clear, **PointData** specifies absolute locations according to the **C** flag.

Note: If this flag is set, the **C** flag (above) is undefined and MUST be ignored.<24>

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record. At least 3 points MUST be specified.

Value	Meaning
0x0000001C ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **Size** is computed as follows: `Size = ((((Count * 0x00000002) + 0x00000014 + 0x00000003) / 4) * 4)`

Value	Meaning
0x00000020 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000004) + 0x00000014`
0x0000002C ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **Size** is computed as follows: `Size = (Count * 0x00000008) + 0x00000014`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data in the record. At least 3 points MUST be specified.

Value	Meaning
0x00000010 ≤ *value*	If the **P** bit is set in the **Flags** field, the minimum **DataSize** is computed as follows: `DataSize = ((((Count * 0x00000002) + 0x0000008 + 0x00000003) / 4) * 4)`
0x00000014 ≤ *value*	If the **P** bit is clear and the **C** bit is set in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000004) + 0x00000008`
0x00000020 ≤ *value*	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, **DataSize** is computed as follows: `DataSize = (Count * 0x00000008) + 0x00000008`

BrushId (4 bytes): A 32-bit unsigned integer that defines the brush, the content of which is determined by the **S** bit in the **Flags** field.

Count (4 bytes): A 32-bit unsigned integer that specifies the number of points in the **PointData** field. At least 3 points MUST be specified.

PointData (variable): An array of **Count** points that define the vertices of the polygon. The first two points in the array specify the first side of the polygon. Each additional point specifies a new side, the vertices of which include the point and the previous point. If the last point and the first point do not coincide, they specify the last side of the polygon.

The type of data in this array is specified by the **Flags** field, as follows:

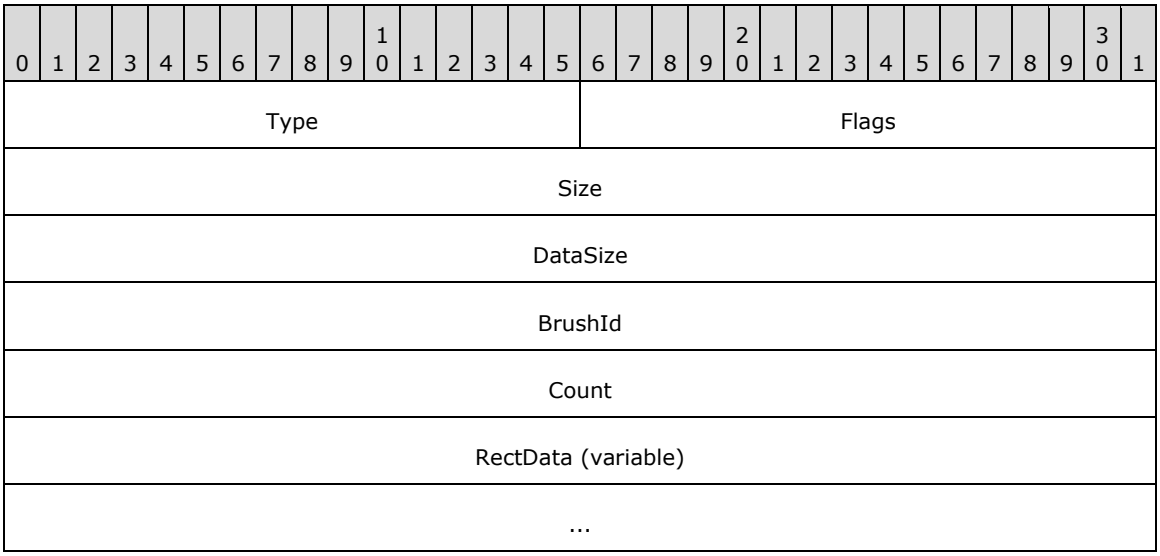
Data Type	Meaning
EmfPlusPointR object	If the **P** flag is set in the **Flags**, the points specify relative locations.
EmfPlusPoint object	If the **P** bit is clear and the **C** bit is set in the **Flags** field, the points specify absolute locations with integer values.
EmfPlusPointF object	If the **P** bit is clear and the **C** bit is clear in the **Flags** field, the points specify

Data Type	Meaning
	absolute locations with floating-point values.

See section 2.3.4 for the specification of additional drawing record types.

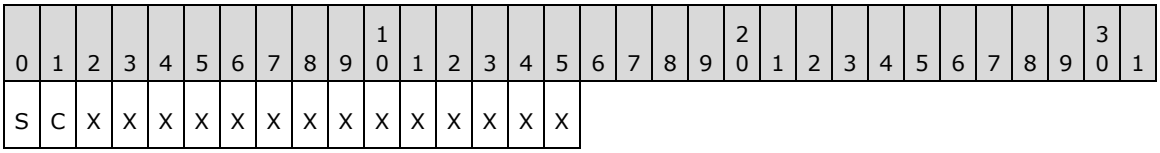
2.3.4.20 EmfPlusFillRects Record

The EmfPlusFillRects record specifies filling the interiors of a series of rectangles.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillRects from the RecordType enumeration. The value MUST be set to 0x400A

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



S (1 bit): This bit specifies the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

C (1 bit): This bit indicates whether the data in the **RectData** field is compressed.

If set, **RectData** contains an EmfPlusRect object. If clear, **RectData** contains an EmfPlusRectF object.

X (1 bit): Reserved and MUST be ignored.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data.

At least 1 **RectData** array element MUST be specified in this record.

Value	Meaning
0x0000001C ≤ *value*	If the **C** bit is set in the **Flags** field, **Size** MUST be computed as follows: Size = (Count * 0x00000008) + 0x00000014
0x00000024 ≤ *value*	If the **C** bit is clear in the **Flags** field, **Size** MUST be computed as follows: Size = (Count * 0x00000010) + 0x00000014

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows.

At least 1 **RectData** array element MUST be specified in this record.

Value	Meaning
0x00000010 ≤ *value*	If the **C** bit is set in the **Flags** field, **DataSize** MUST be computed as follows: `DataSize = (Count * 0x00000008) + 0x00000008`
0x00000018 ≤ *value*	If the **C** bit is clear in the **Flags** field, **DataSize** MUST be computed as follows: `DataSize = (Count * 0x00000010) + 0x00000008`

BrushId (4 bytes): A 32-bit unsigned integer that defines the brush, the content of which is determined by the **S** bit in the **Flags** field.

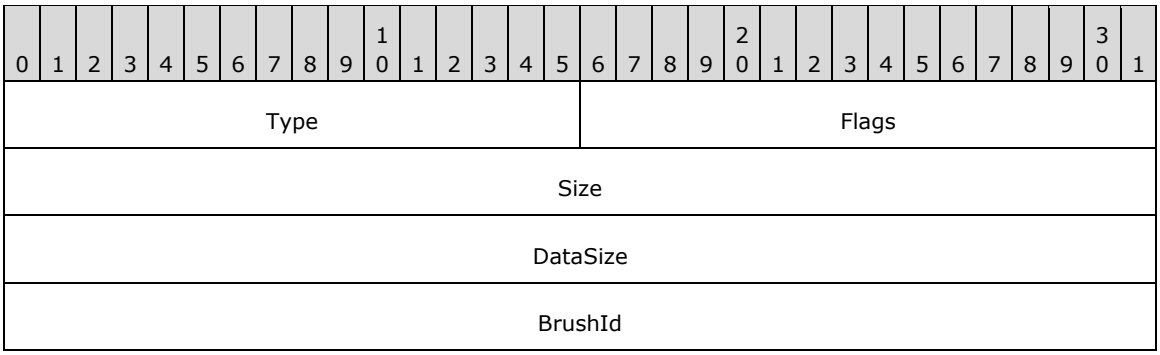
Count (4 bytes): A 32-bit unsigned integer that specifies the number of rectangles in the **RectData** field.

RectData (variable): An array of either an EmfPlusRect or EmfPlusRectF objects of **Count** length that defines the rectangle data.

See section 2.3.4 for the specification of additional drawing record types.

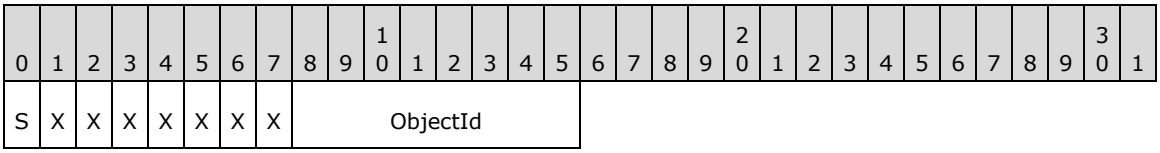
2.3.4.21 EmfPlusFillRegion Record

The EmfPlusFillRegion record specifies filling the interior of a graphics region.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusFillRegion from the RecordType enumeration. The value MUST be 0x4013

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



S (1 bit): This bit specifies the type of data in the **BrushId** field.

If set, **BrushId** specifies a color as an EmfPlusARGB object. If clear, **BrushId** contains the index of an EmfPlusBrush object in the EMF+ Object Table.

X (1 bit): Reserved and MUST be ignored.

ObjectId (1 byte): The index of the EmfPlusRegion object to fill, in the EMF+ Object Table. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000004.

BrushId (4 bytes): A 32-bit unsigned integer that defines the brush, the content of which is determined by the **S** bit in the **Flags** field.

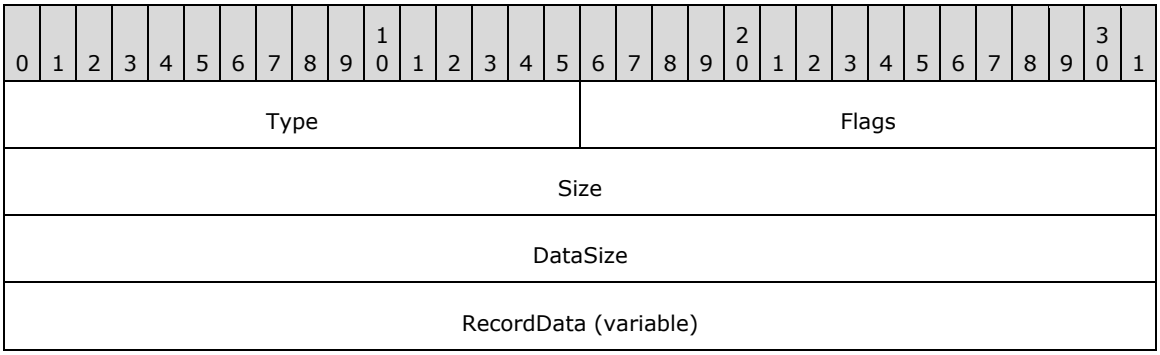
See section 2.3.4 for the specification of additional drawing record types.

2.3.5 Object Record Types

The Object Record Types define reusable graphics objects. The following are EMF+ object record types:

Name	Section	Description
EmfPlusObject	2.3.5.1	Defines an object for use in graphics operations.
EmfPlusSerializableObject	2.3.5.2	Defines an object that has been serialized into a data buffer.

The generic structure of EMF+ object records is specified as follows:



...

Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The object record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusObject	0x4008
EmfPlusSerializableObject	0x4038

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, MUST define the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

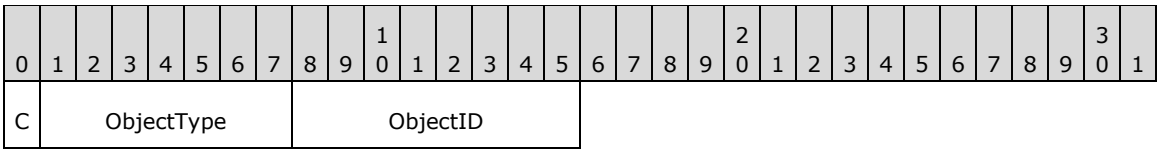
2.3.5.1 EmfPlusObject Record

The EmfPlusObject record specifies an object for use in graphics operations. The object definition can span multiple records, which is indicated by the value of the **Flags** field.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type																Flags															
Size																															
TotalObjectSize (optional)																															
DataSize																															
ObjectData (variable)																															
...																															

Type (2 bytes): A 16-bit unsigned integer that defines this record type as EmfPlusObject from the RecordType enumeration. The value MUST be 0x4008.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



C (1 bit): Indicates that the object definition continues on in the next EmfPlusObject record. This flag is never set in the final record that defines the object.

ObjectType (7 bits): Specifies the type of object to be created by this record, from the ObjectType enumeration.

ObjectID (1 byte): The index in the EMF+ Object Table to associate with the object created by this record. The value MUST be zero to 63, inclusive.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

TotalObjectSize (4 bytes): If the record is continuable, when the continue bit is set, this field will be present. Continuing objects have multiple EMF+ records starting with EmfPlusContineudObjectRecord. Each EmfPlusContinuedObjectRecord will contain a **TotalObjectSize**. Once **TotalObjectSize** number of bytes has been read, the next EMF+ record will not be treated as part of the continuing object.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value varies based on the size of object.

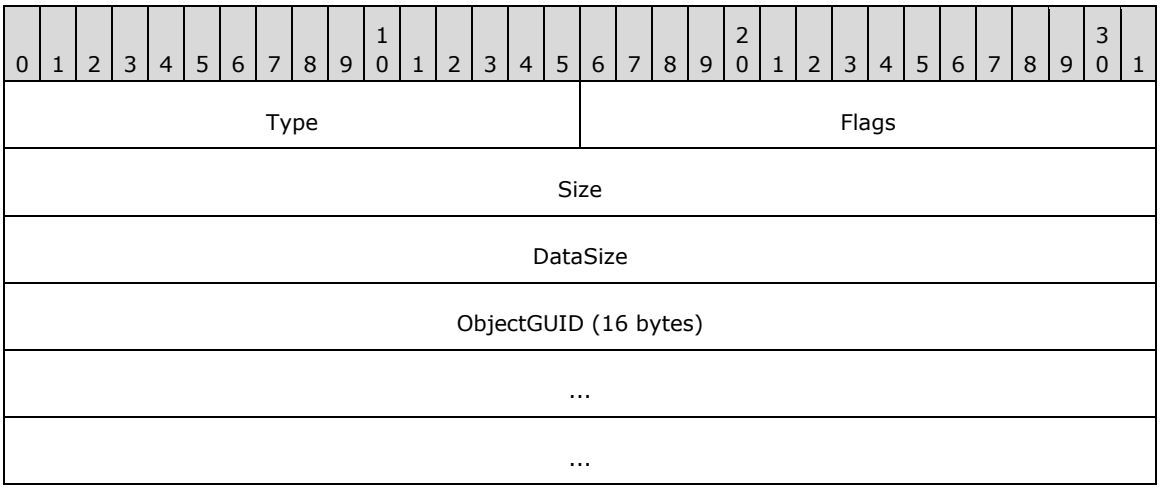
ObjectData (variable): An array of bytes that contains data for the type of object specified in the **Flags** field. The content and format of the data can be different for each object type. See the individual object definitions in section 2.2.1 for additional information.

The EmfPlusObject record is generic; it is used for all types of objects. Values that are specific to particular object types are contained in the **ObjectData** field. A conceptual model for managing graphics objects is described in Managing Graphics Objects.

See section 2.3.5 for the specification of additional object record types.

2.3.5.2 EmfPlusSerializableObject Record

The EmfPlusSerializableObject record defines an image effects parameter block that has been serialized into a data buffer.<25>



BufferSize
Buffer (variable)
...

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSerializableObject from the RecordType enumeration. The value MUST be 0x4038.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be computed as follows:

```
Size = BufferSize + 0x00000020
```

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be computed as follows:

```
DataSize = BufferSize + 0x00000014
```

ObjectGUID (16 bytes): The GUID packet representation value ([MS-DTYP] section 2.3.4.2) for the image effect. This MUST correspond to one of the ImageEffects identifiers.

BufferSize (4 bytes): A 32-bit unsigned integer that specifies the size in bytes of the 32-bit-aligned **Buffer** field.

Buffer (variable): An array of **BufferSize** bytes that contain the serialized image effects parameter block that corresponds to the GUID in the **ObjectGUID** field. This MUST be one of the Image Effects objects.

See section 2.3.5 for the specification of additional object record types.

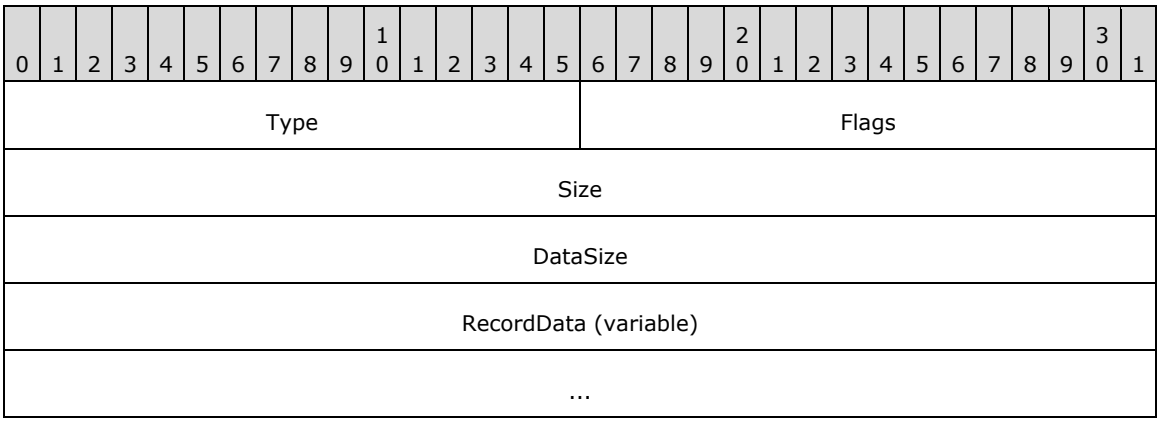
2.3.6 Property Record Types

The Property Record Types specify properties of the playback device context. The following are EMF+ property record types:

Name	Section	Description
EmfPlusSetAntiAliasMode	2.3.6.1	Specifies the anti-aliasing mode for text output.
EmfPlusSetCompositingMode	2.3.6.2	Specifies how source colors are combined with background colors.
EmfPlusSetCompositingQuality	2.3.6.3	Specifies the desired level of quality for creating composite images from multiple objects.
EmfPlusSetInterpolationMode	2.3.6.4	Specifies how image scaling, including stretching and shrinking, is performed.
EmfPlusSetPixelOffsetMode	2.3.6.5	Specifies how pixels are centered with respect to the coordinates of the drawing surface.

Name	Section	Description
EmfPlusSetRenderingOrigin	2.3.6.6	Specifies the rendering origin for graphics output.
EmfPlusSetTextContrast	2.3.6.7	Specifies text contrast according to the gamma correction value.
EmfPlusSetTextRenderingHint	2.3.6.8	Specifies the quality of text rendering, including the type of anti-aliasing.

The generic structure of EMF+ property records is specified as follows:



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The property record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusSetRenderingOrigin	0x401D
EmfPlusSetAntiAliasMode	0x401E
EmfPlusSetTextRenderingHint	0x401F
EmfPlusSetTextContrast	0x4020
EmfPlusSetInterpolationMode	0x4021
EmfPlusSetPixelOffsetMode	0x4022
EmfPlusSetCompositingMode	0x4023
EmfPlusSetCompositingQuality	0x4024

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

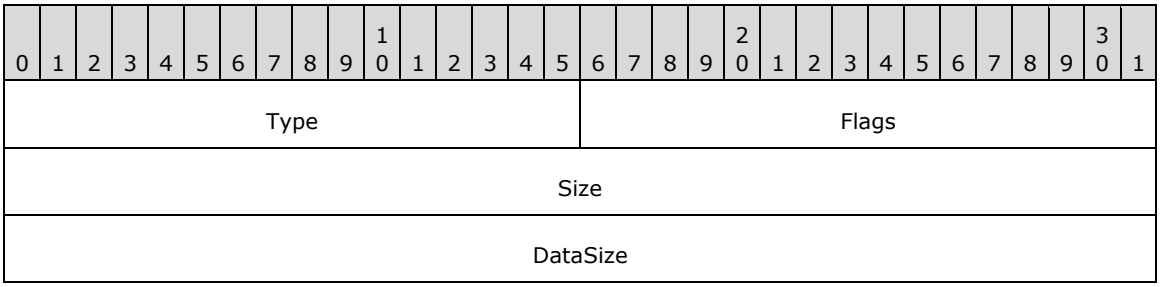
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, MUST define the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

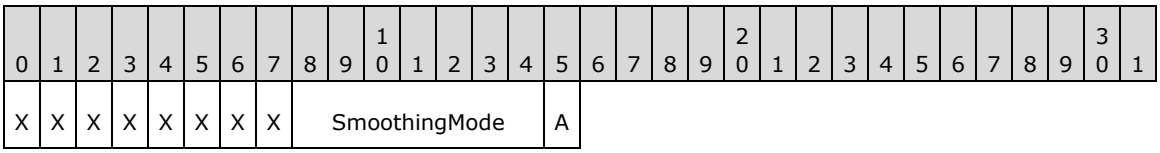
2.3.6.1 EmfPlusSetAntiAliasMode Record

The EmfPlusSetAntiAliasMode record specifies the anti-aliasing mode for text output.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetAntiAliasMode from the RecordType enumeration. The value MUST be 0x401E.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

SmoothingMode (7 bits): The smoothing mode value, from the SmoothingMode enumeration.<26>

A (1 bit): If set, anti-aliasing SHOULD be performed.

If clear, anti-aliasing SHOULD NOT be performed.

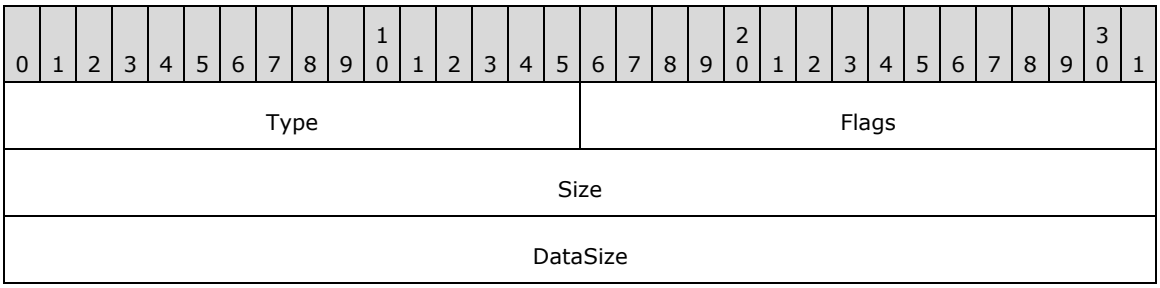
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of data in the record-specific data that follows. This number does not include the size of the invariant part of this record. For this record type, the value MUST be 0x00000000.

See section 2.3.6 for the specification of additional property record types.

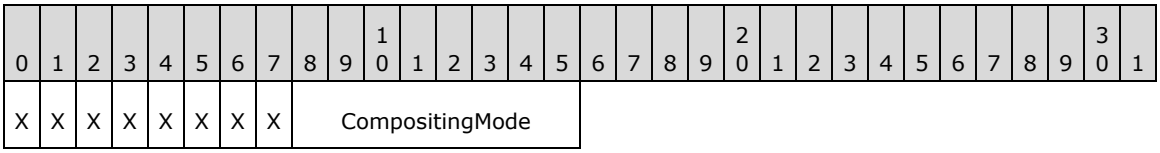
2.3.6.2 EmfPlusSetCompositingMode Record

The EmfPlusSetCompositingMode record specifies how source colors are combined with background colors.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetCompositingMode from the RecordType enumeration. The value MUST be 0x4023.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

CompositingMode (1 byte): The compositing mode value, from the CompositingMode enumeration. Compositing can be expressed as the state of alpha blending, which can either be on or off.

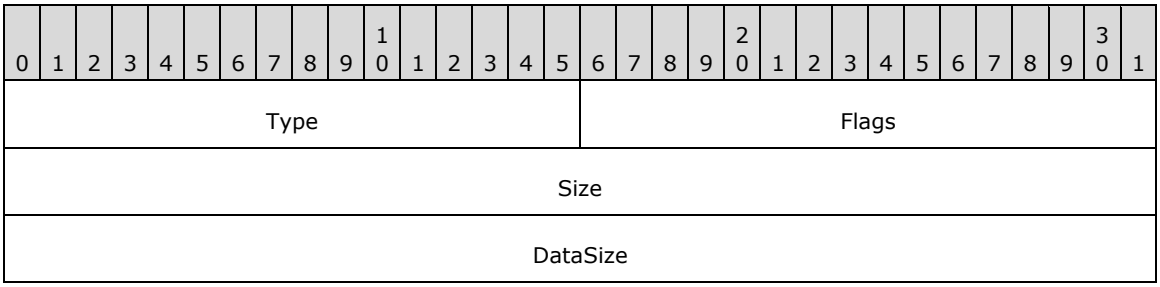
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.6 for the specification of additional property record types.

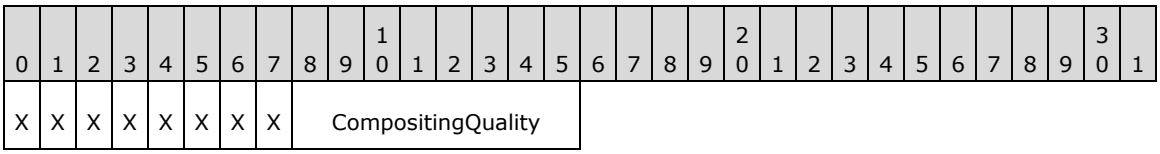
2.3.6.3 EmfPlusSetCompositingQuality Record

The EmfPlusSetCompositingQuality record specifies the desired level of quality for creating composite images from multiple objects.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetCompositingQuality from the RecordType enumeration. The value MUST be 0x4024.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

CompositingQuality (1 byte): The compositing quality value, from the CompositingQuality enumeration.

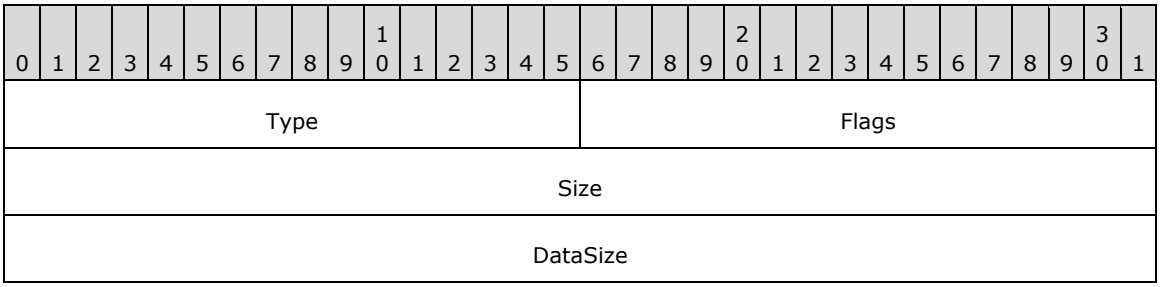
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.6 for the specification of additional property record types.

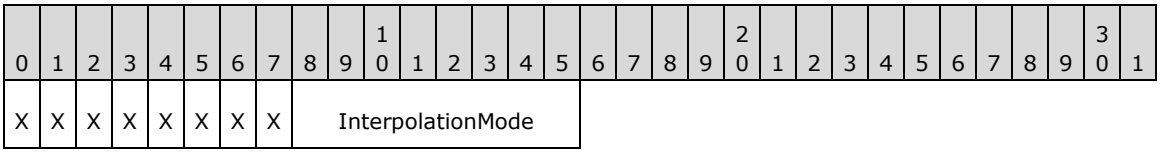
2.3.6.4 EmfPlusSetInterpolationMode Record

The EmfPlusSetInterpolationMode record specifies how image scaling, including stretching and shrinking, is performed.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetInterpolationMode from the RecordType enumeration. The value MUST be 0x4021.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

InterpolationMode (1 byte): The interpolation mode value, from the InterpolationMode enumeration.

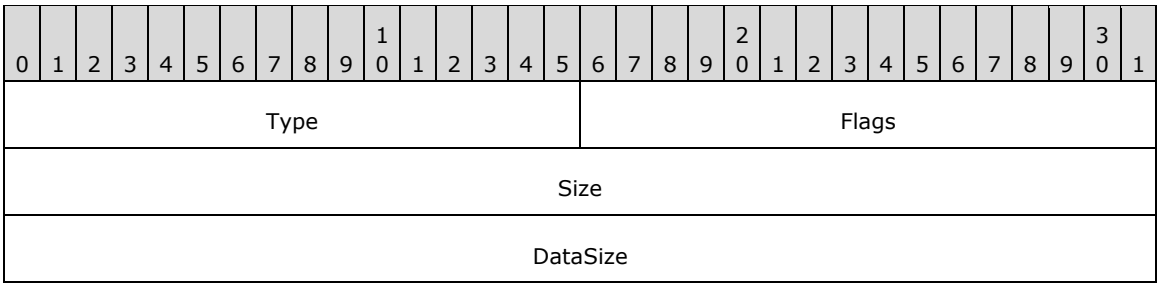
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.6 for the specification of additional property record types.

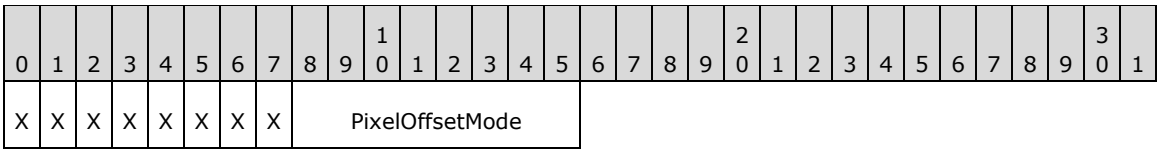
2.3.6.5 EmfPlusSetPixelOffsetMode Record

The EmfPlusSetPixelOffsetMode record specifies how pixels are centered with respect to the coordinates of the drawing surface.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetPixelOffsetMode from the RecordType enumeration. The value MUST be 0x4022.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

PixelOffsetMode (1 byte): The pixel offset mode value, from the PixelOffsetMode enumeration.

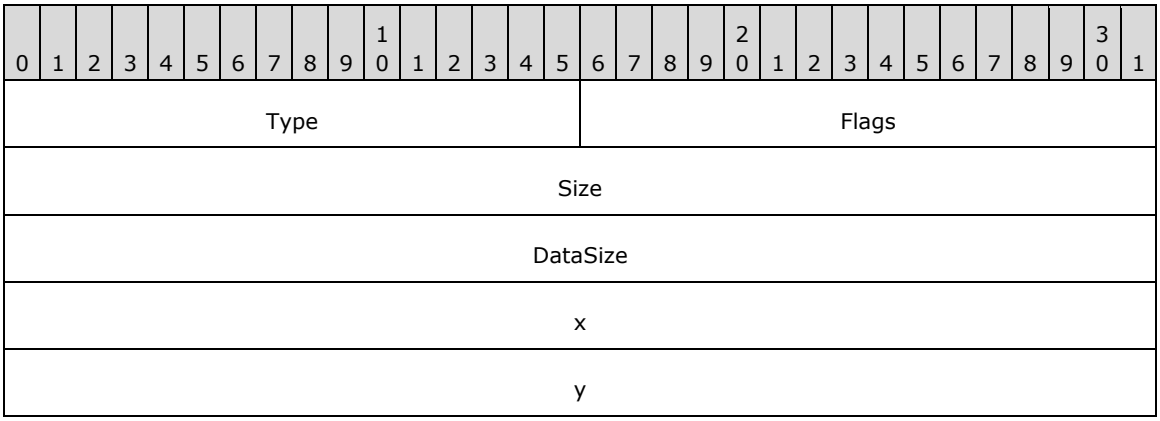
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.6 for the specification of additional property record types.

2.3.6.6 EmfPlusSetRenderingOrigin Record

The EmfPlusSetRenderingOrigin record specifies the rendering origin for graphics output.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetRenderingOrigin from the RecordType enumeration. The value MUST be 0x401D.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000014.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000008.

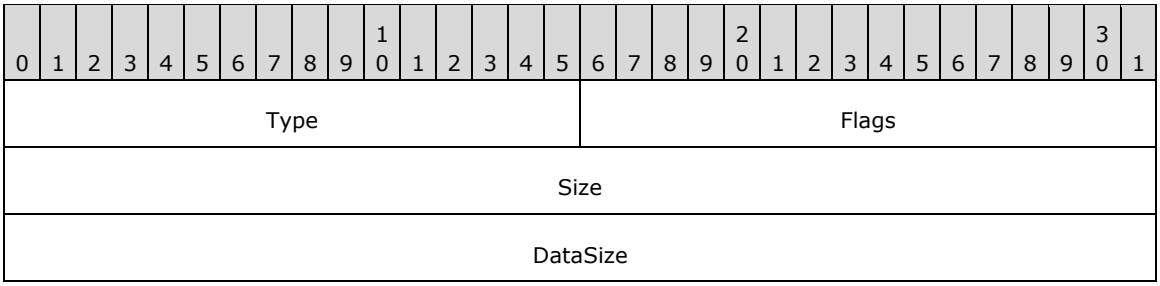
x (4 bytes): A 32-bit unsigned integer that defines the horizontal coordinate value of the rendering origin.

y (4 bytes): A 32-bit unsigned integer that defines the vertical coordinate value of the rendering origin.

See section 2.3.6 for the specification of additional property record types.

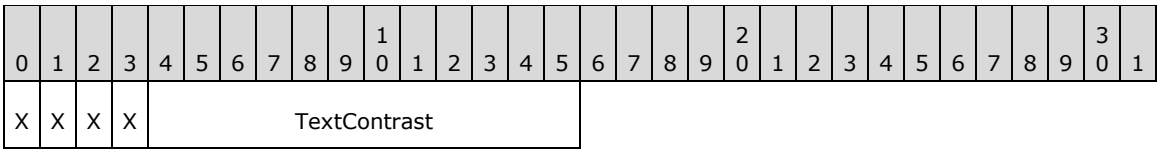
2.3.6.7 EmfPlusSetTextContrast Record

The EmfPlusSetTextContrast record specifies text contrast according to the gamma correction value.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetTextContrast from the RecordType enumeration. The value MUST be 0x4020.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

TextContrast (12 bits): The gamma correction value X 1000, which will be applied to subsequent text rendering operations. The allowable range is 1000 to 2200, representing text gamma values of 1.0 to 2.2.

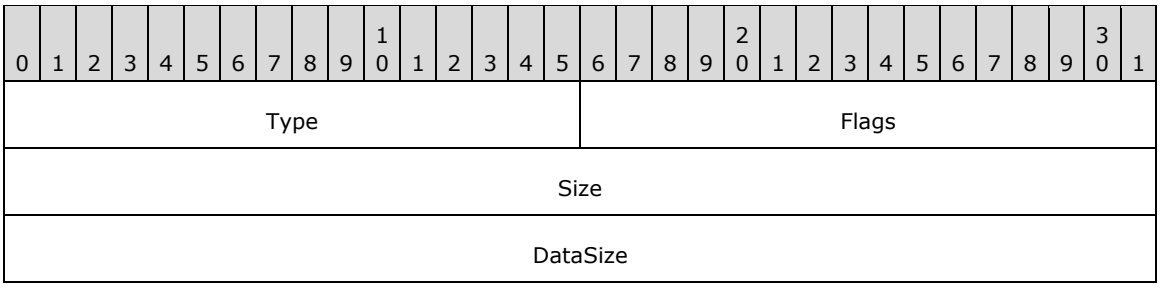
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.6 for the specification of additional property record types.

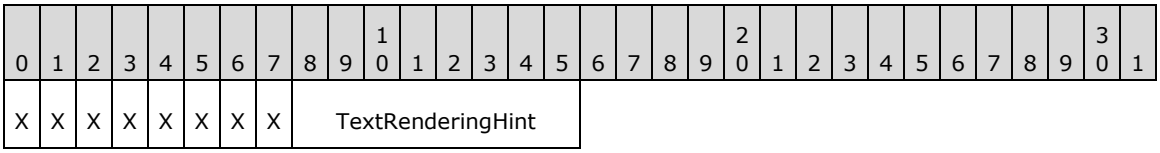
2.3.6.8 EmfPlusSetTextRenderingHint Record

The EmfPlusSetTextRenderingHint record specifies the quality of text rendering, including the type of anti-aliasing.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetTextRenderingHint from the RecordType enumeration. The value MUST be 0x401F.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

TextRenderingHint (1 byte): The text rendering hint value, from the TextRenderingHint enumeration, which specifies the quality to use in subsequent text rendering.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

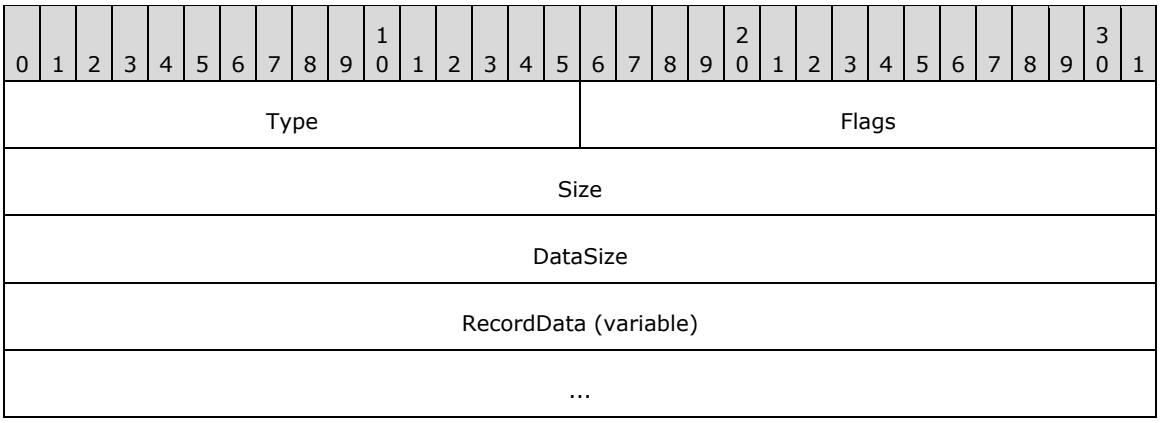
See section 2.3.6 for the specification of additional property record types.

2.3.7 State Record Types

The State Record Types specify operations on the state of the playback device context. The following are EMF+ state record types:

Name	Section	Description
EmfPlusBeginContainer	2.3.7.1	Starts a new graphics state container with a transformation.
EmfPlusBeginContainerNoParams	2.3.7.2	Starts a new graphics state container.
EmfPlusEndContainer	2.3.7.3	Closes a graphics state container that was previously opened by a begin container operation.
EmfPlusRestore	2.3.7.4	Restores a saved graphics state.
EmfPlusSave	2.3.7.5	Saves the current graphics state.

The generic structure of EMF+ state records is specified as follows:



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The state record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusSave	0x4025
EmfPlusRestore	0x4026
EmfPlusBeginContainer	0x4027
EmfPlusBeginContainerNoParams	0x4028
EmfPlusEndContainer	0x4029

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

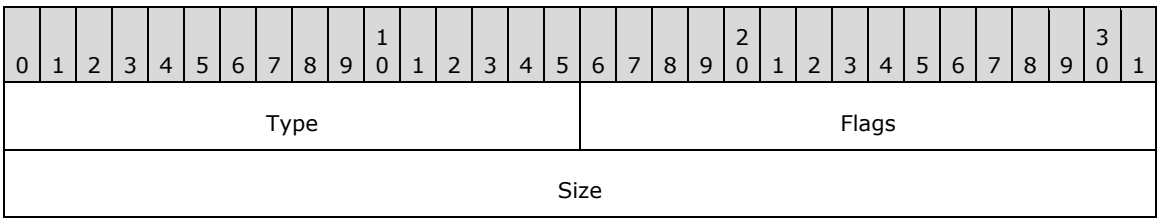
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, MUST define the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

2.3.7.1 EmfPlusBeginContainer Record

The EmfPlusBeginContainer record opens a new graphics state container and specifies a transform for it.



DataSize
DestRect (16 bytes)
...
...
SrcRect (16 bytes)
...
...
StackIndex

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusBeginContainer from the RecordType enumeration. The value MUST be 0x4027.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of 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
PageUnit								0	0	0	0	0	0	0	0																

PageUnit (1 byte): The unit of measure for **page space** coordinates, from the UnitType enumeration. This value SHOULD NOT be **UnitTypeDisplay** or **UnitTypeWorld**.<27>

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000030.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000024.

DestRect (16 bytes): An EmfPlusRectF object that, with **SrcRect**, specifies a transform for the container. This transformation results in **SrcRect** when applied to **DestRect**.

SrcRect (16 bytes): An EmfPlusRectF rectangle that, with **DestRect**, specifies a transform for the container. This transformation results in **SrcRect** when applied to **DestRect**.

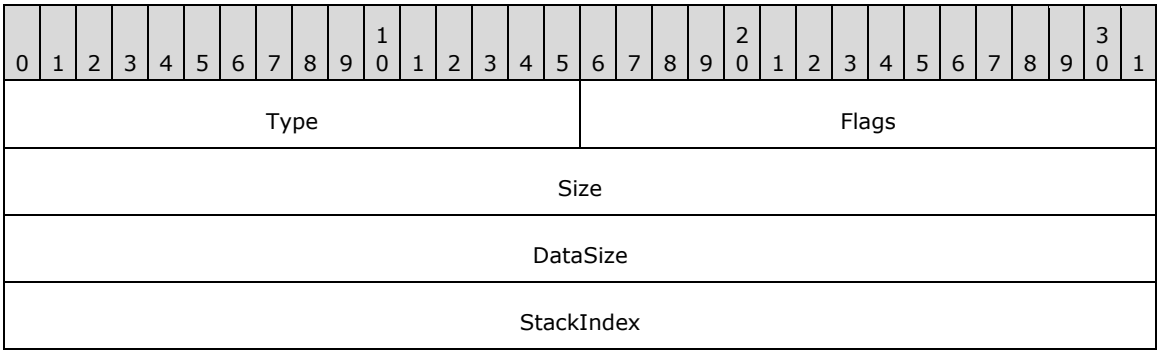
StackIndex (4 bytes): A 32-bit unsigned integer that specifies an index to associate with the graphics state container. The index MUST be referenced by a subsequent EmfPlusEndContainer to close the graphics state container.

Each graphics state container MUST be added to an array of saved graphics containers. The graphics state container is not written to the EMF+ metafile, so its format can be determined by the implementation.

See section 2.3.7 for the specification of additional state record types.

2.3.7.2 EmfPlusBeginContainerNoParams Record

The EmfPlusBeginContainerNoParams record opens a new graphics state container.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusBeginContainerNoParams from the RecordType enumeration. The value MUST be 0x4028.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000004.

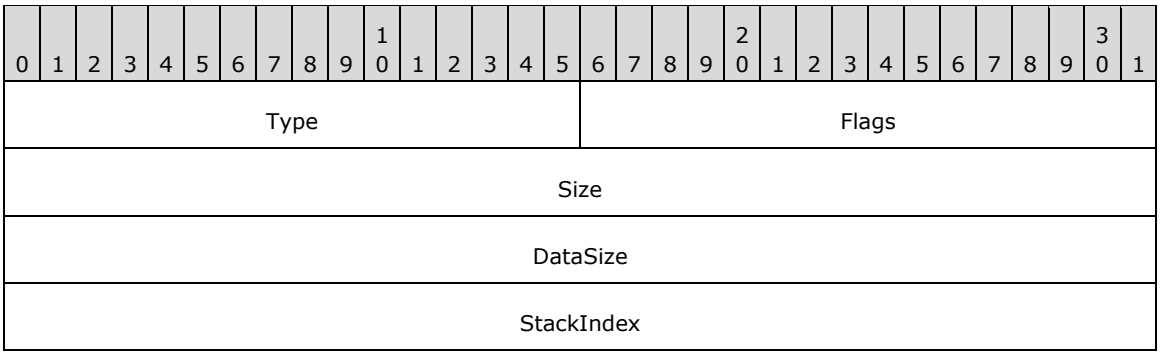
StackIndex (4 bytes): A 32-bit unsigned integer that specifies an index to associate with the graphics state container. The index MUST be referenced by a subsequent EmfPlusEndContainer record to close the graphics state container.

Each graphics state container MUST be added to an array of saved graphics containers. The graphics state container is not written to the EMF+ metafile, so its format can be determined by the implementation.

See section 2.3.7 for the specification of additional state record types.

2.3.7.3 EmfPlusEndContainer Record

The EmfPlusEndContainer record closes a graphics state container that was previously opened by a begin container operation.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusEndContainer from the RecordType enumeration. The value MUST be 0x4029.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, this value is 0x00000010.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, this value is 0x00000004.

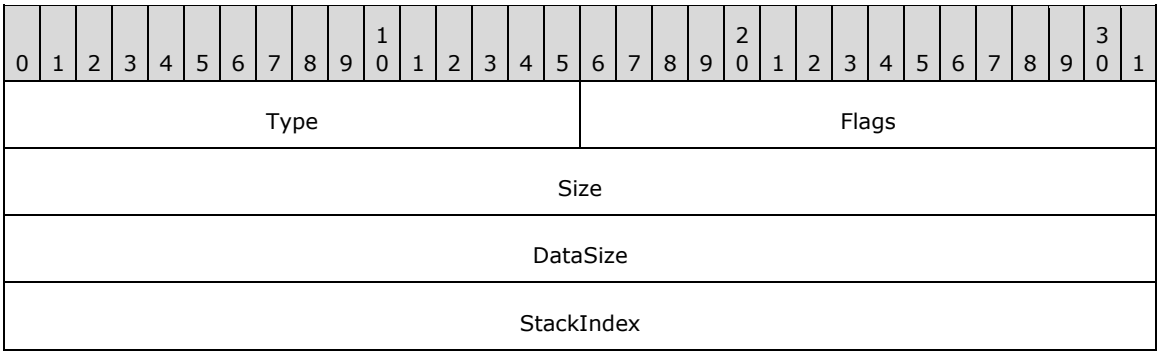
StackIndex (4 bytes): A 32-bit unsigned integer that specifies the index of a graphics state container. The index MUST match the value associated with a graphics state container opened by a previous EmfPlusBeginContainer or EmfPlusBeginContainerNoParams record.

Each graphics state container MUST be added to an array of saved graphics containers. The graphics state container is not written to the EMF+ metafile, so its format can be determined by the implementation.

See section 2.3.7 for the specification of additional state record types.

2.3.7.4 EmfPlusRestore Record

The EmfPlusRestore record restores the graphics state, identified by a specified index, from a stack of saved graphics states.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusRestore from the RecordType enumeration. The value MUST be 0x4026.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of records in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000004.

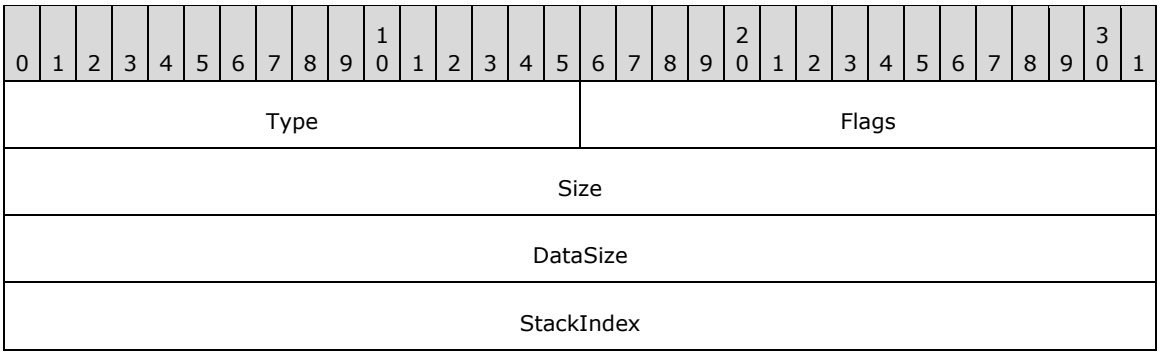
StackIndex (4 bytes): A 32-bit unsigned integer that specifies the level associated with a graphics state. The level value was assigned to the graphics state by a previous EmfPlusSave record.

Each graphics state MUST be popped off a stack of saved graphics states. The graphics state information is not written to the EMF+ metafile, so its format can be determined by the implementation.

See section 2.3.7 for the specification of additional state record types.

2.3.7.5 EmfPlusSave Record

The EmfPlusSave record saves the graphics state, identified by a specified index, on a stack of saved graphics states.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as **EmfPlusSave** from the RecordType enumeration. The value MUST be 0x4025.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of records in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000004.

StackIndex (4 bytes): A 32-bit unsigned integer that specifies a level to associate with the graphics state. The level value can be used by a subsequent EmfPlusRestore record to retrieve the graphics state.

Each saved graphics state MUST be pushed onto a stack of saved graphics states. The graphics state information is not written to the EMF+ metafile, so its format can be determined by the implementation.

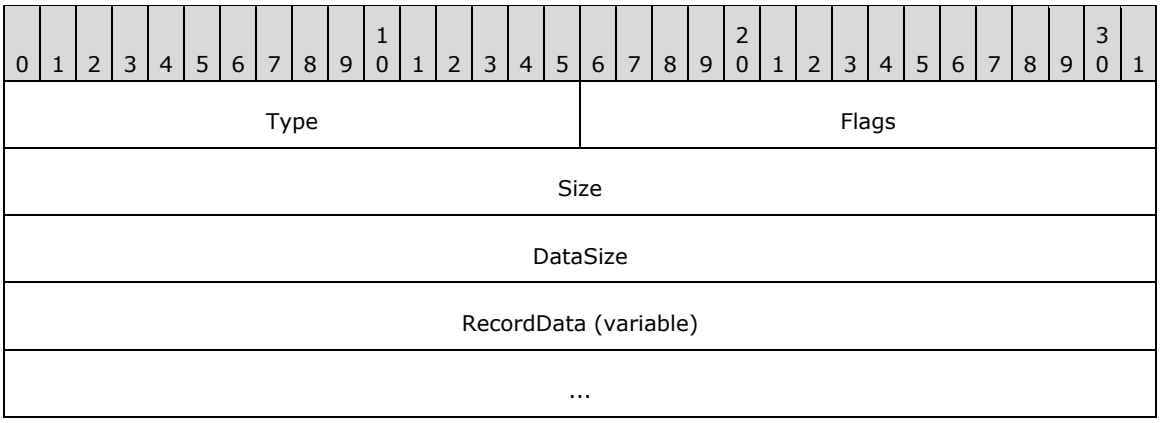
See section 2.3.7 for the specification of additional state record types.

2.3.8 Terminal Server Record Types

The Terminal Server Record Types specify graphics processing on a terminal server. The following are EMF+ terminal server record types.

Name	Section	Description
EmfPlusSetTSClip	2.3.8.1	Specifies clipping areas in the graphics device context for a terminal server.<28>
EmfPlusSetTSGraphics	2.3.8.2	Specifies the state of a graphics device context for a terminal server.<29>

The generic structure of EMF+ terminal server records is specified as follows.



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The terminal server types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusSetTSGraphics	0x4039
EmfPlusSetTSClip	0x403A

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

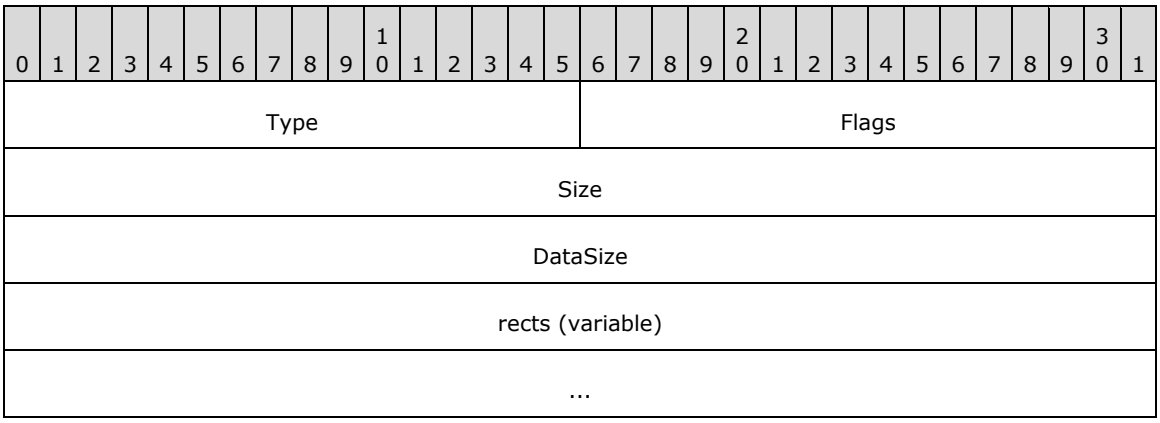
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, MUST define the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

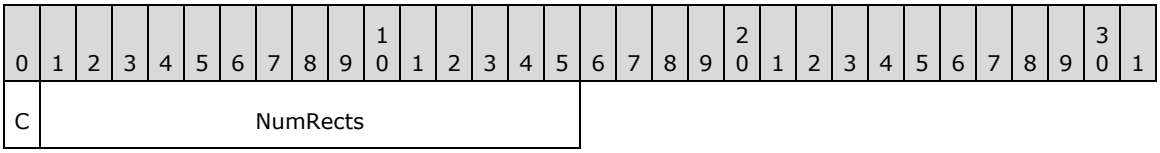
2.3.8.1 EmfPlusSetTSClip Record

The EmfPlusSetTSClip record specifies clipping areas in the graphics device context for a terminal server.<30>



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetTSClip from the RecordType enumeration. The value MUST be 0x403A.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



C (1 bit): This bit specifies the format of the rectangle data in the **rects** field. If set, each rectangle is defined in 4 bytes. If clear, each rectangle is defined in 8 bytes.

NumRects (15 bits): This field specifies the number of rectangles that are defined in the **rect** field.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. The computation of this value is determined by the **C** bit in the **Flags** field, as shown in the following table.

C bit value	Meaning
0	**NumRects** rectangles, consisting of 8 bytes each, are defined in the **rects** field, and **Size** is computed as follows: `Size = (NumRects * 0x00000008) + 0x0000000C`
1	**NumRects** rectangles, consisting of 4 bytes each, are defined in the **rects** field, and **Size** is computed as follows: `Size = (NumRects * 0x00000004) + 0x0000000C`

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. The computation of this value is determined by the **C** bit in the **Flags** field, as shown in the following table.

C bit value	Meaning
0	**NumRects** rectangles, consisting of 8 bytes each, are defined in the **rects** field, and **DataSize** is computed as follows: `DataSize = NumRects * 0x00000008`
1	**NumRects** rectangles, consisting of 4 bytes each, are defined in the **rects** field, and **DataSize** is computed as follows: `DataSize = NumRects * 0x00000004`

rects (variable): An array of **NumRects** rectangles that define clipping areas. The format of this data is determined by the **C** bit in the **Flags** field.

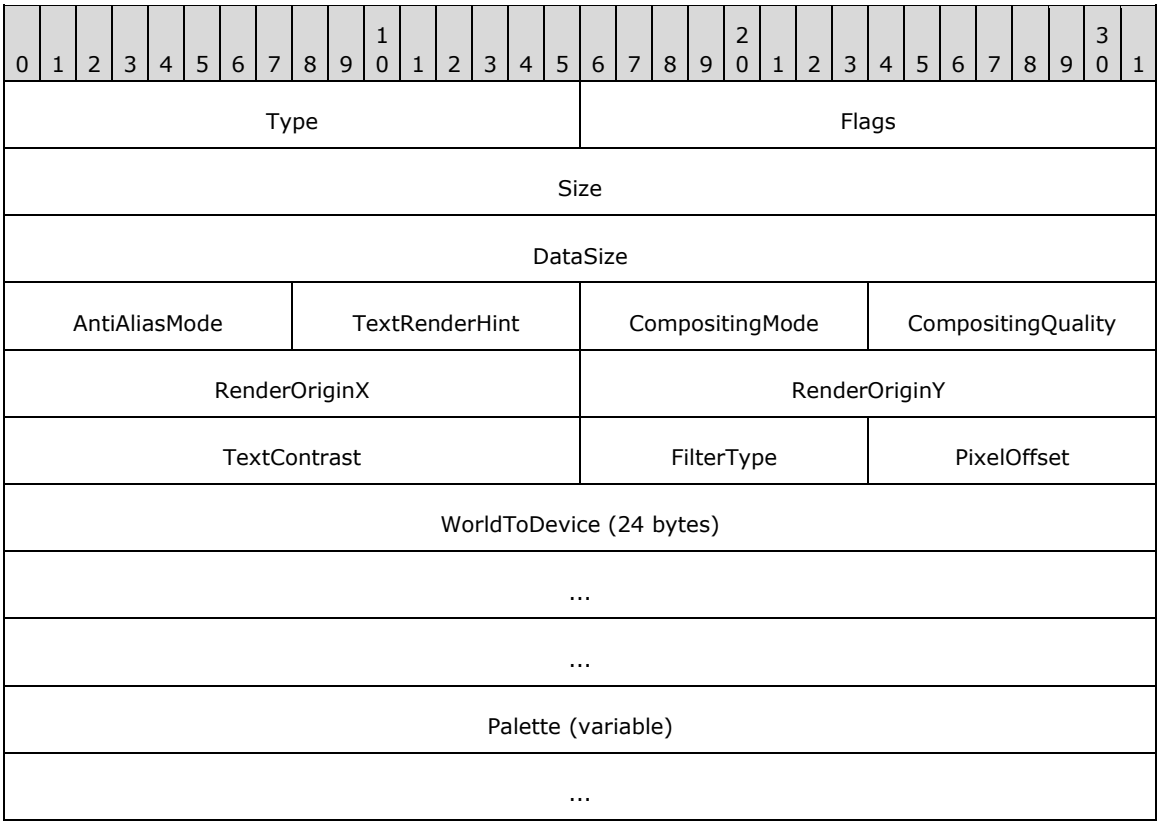
The compression scheme for data in this record uses the following algorithm. Each point of each rectangle is encoded in either a single byte or 2 bytes. If the point is encoded in a single byte, the high bit (0x80) of the byte MUST be set, and the value is a signed number represented by the lower 7 bits. If the high bit is not set, then the value is encoded in 2 bytes, with the high-order byte encoded in the 7 lower bits of the first byte, and the low-order byte value encoded in the second byte.

Each point is encoded as the difference between the point in the current rect and the point in the previous rect. The bottom point of the rect is encoded as the difference between the bottom coordinate and the top coordinate on the current rect.

See section 2.3.8 for the specification of additional terminal server record types.

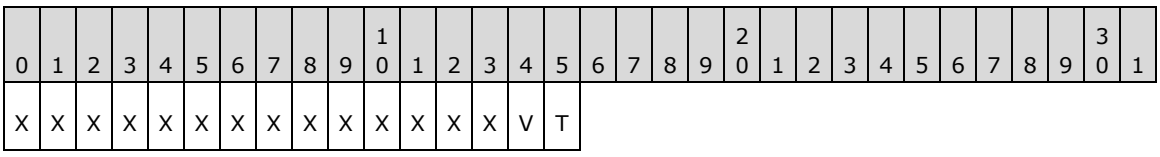
2.3.8.2 EmfPlusSetTSGraphics Record

The EmfPlusSetTSGraphics record specifies the state of a graphics device context for a terminal server. <31>



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetTSGraphics from the RecordType enumeration. The value MUST be 0x4039.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

V (1 bit): If set, the palette contains only the basic VGA colors.

T (1 bit): If set, this record contains an EmfPlusPalette object in the **Palette** field following the graphics state data.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and record-specific data. This value MUST be 0x00000030 plus the size of the **Palette** field.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. This value MUST be 0x00000024 plus the size of the **Palette** field.

AntiAliasMode (1 byte): An 8-bit unsigned integer that specifies the quality of line rendering, including the type of line anti-aliasing. It MUST be defined in the SmoothingMode enumeration.

TextRenderHint (1 byte): An 8-bit unsigned integer that specifies the quality of text rendering, including the type of text anti-aliasing. It MUST be defined in the TextRenderingHint enumeration.

CompositingMode (1 byte): An 8-bit unsigned integer that specifies how source colors are combined with background colors. It MUST be a value in the CompositingMode enumeration.

CompositingQuality (1 byte): An 8-bit unsigned integer that specifies the degree of smoothing to apply to lines, curves and the edges of filled areas to make them appear more continuous or sharply defined. It MUST be a value in the CompositingQuality enumeration.

RenderOriginX (2 bytes): A 16-bit signed integer, which is the horizontal coordinate of the origin for rendering halftoning and **dithering** matrixes.

RenderOriginY (2 bytes): A 16-bit signed integer, which is the vertical coordinate of the origin for rendering halftoning and dithering matrixes.

TextContrast (2 bytes): A 16-bit unsigned integer that specifies the gamma correction value used for rendering anti-aliased and ClearType text. This value MUST be in the range 0 to 12, inclusive.

FilterType (1 byte): An 8-bit unsigned integer that specifies how scaling, including stretching and shrinking, is performed. It MUST be a value in the FilterType enumeration.

PixelOffset (1 byte): An 8-bit unsigned integer that specifies the overall quality of the image and text-rendering process. It MUST be a value in the PixelOffsetMode enumeration.

WorldToDevice (24 bytes): An 192-bit EmfPlusTransformMatrix object that specifies the world space to device space transforms.

Palette (variable): An optional EmfPlusPalette object.

See section 2.3.8 for the specification of additional terminal server record types.

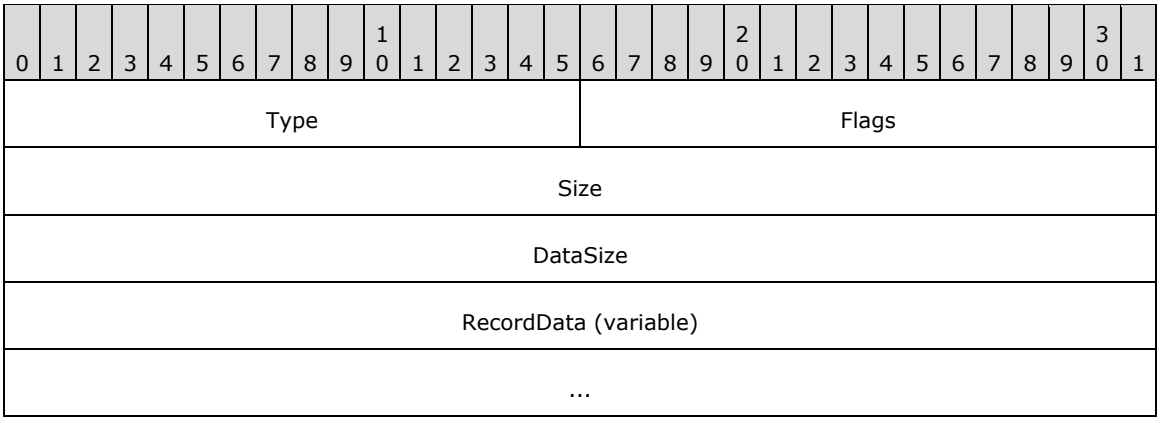
2.3.9 Transform Record Types

The Transform Record Types specify properties and transforms on coordinate spaces. The following are EMF+ transform record types:

Name	Section	Description
EmfPlusMultiplyWorldTransform	2.3.9.1	Multiplies the current world space transform by a specified transform matrix.
EmfPlusResetWorldTransform	2.3.9.2	Resets the current world space transform to the identify matrix.

Name	Section	Description
EmfPlusRotateWorldTransform	2.3.9.3	Performs a rotation on the current world space transform.
EmfPlusScaleWorldTransform	2.3.9.4	Performs a scaling on the current world space transform.
EmfPlusSetPageTransform	2.3.9.5	Specifies scaling factors and units for converting page space coordinates to device space coordinates.
EmfPlusSetWorldTransform	2.3.9.6	Sets the current world space transform according to the values in a specified transform matrix.
EmfPlusTranslateWorldTransform	2.3.9.7	Performs a translation on the current world space transform.

The generic structure of EMF+ transform records is specified as follows:



Type (2 bytes): A 16-bit unsigned integer that identifies the record type. The transform record types are listed below. See the table above for descriptions of these records.

Name	Value
EmfPlusSetWorldTransform	0x402A
EmfPlusResetWorldTransform	0x402B
EmfPlusMultiplyWorldTransform	0x402C
EmfPlusTranslateWorldTransform	0x402D
EmfPlusScaleWorldTransform	0x402E
EmfPlusRotateWorldTransform	0x402F
EmfPlusSetPageTransform	0x4030

Flags (2 bytes): A 16-bit unsigned integer that contains information for some records on how the operation is to be performed and on the structure of the record.

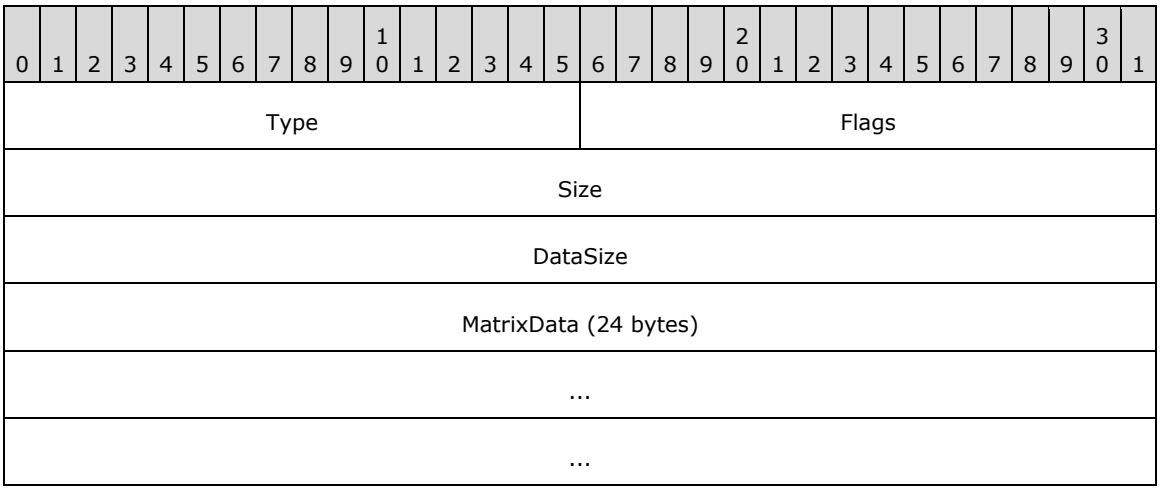
Size (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific data.

DataSize (4 bytes): A 32-bit unsigned integer that MUST define the 32-bit-aligned number of bytes of data in the **RecordData** field that follows. This number does not include the 12-byte record header.

RecordData (variable): An optional, variable-length array of bytes that, if present, MUST define the data specific to individual records. For specifications of the additional information, if any, which is contained within this field, see individual record definitions.

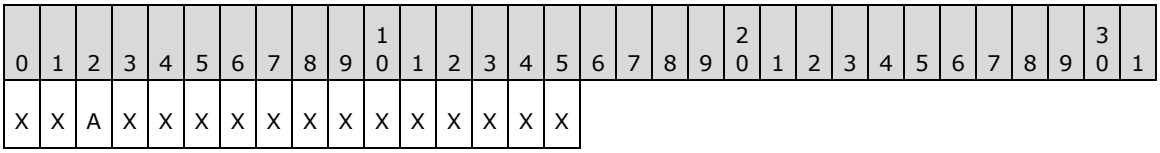
2.3.9.1 EmfPlusMultiplyWorldTransform Record

The EmfPlusMultiplyWorldTransform record multiplies the current world space transform by a specified transform matrix.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusMultiplyWorldTransform from the RecordType enumeration. The value MUST be 0x402C.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

A (1 bit): If set, the transform matrix is post-multiplied. If clear, it is pre-multiplied.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, this value MUST be 0x00000024.

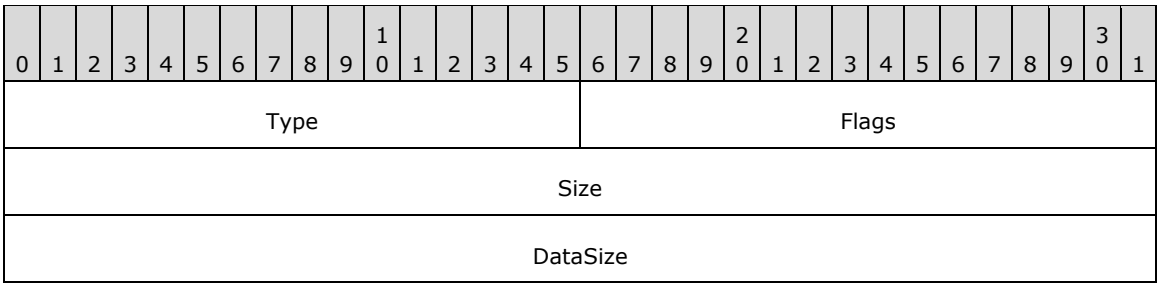
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data. For this record type, this value MUST be 0x00000018.

MatrixData (24 bytes): An EmfPlusTransformMatrix object that defines the multiplication matrix.

See section 2.3.9 for the specification of additional transform record types.

2.3.9.2 EmfPlusResetWorldTransform Record

The EmfPlusResetWorldTransform record resets the current world space transform to the identify matrix.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusResetWorldTransform from the RecordType enumeration. The value MUST be 0x402B.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

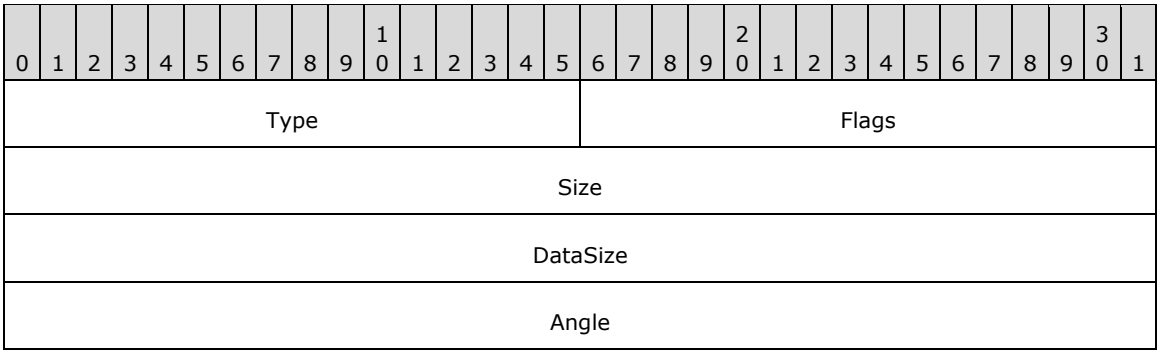
Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x0000000C.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000000.

See section 2.3.9 for the specification of additional transform record types.

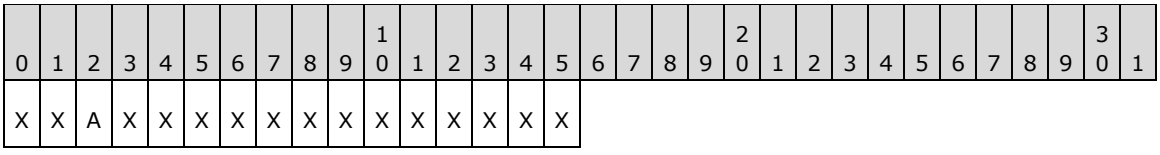
2.3.9.3 EmfPlusRotateWorldTransform Record

The EmfPlusRotateWorldTransform record performs a rotation on the current world space transform.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusRotateWorldTransform from the RecordType enumeration. The value MUST be 0x402F.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

A (1 bit): If set, the transform matrix is post-multiplied. If clear, it is pre-multiplied.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000004.

Angle (4 bytes): A 32-bit floating-point value that specifies the angle of rotation in degrees. The operation is performed by constructing a new transform matrix from the following diagram.

sin(Angle)	cos(Angle)	0
cos(Angle)	-sin(Angle)	0

Figure 3: Rotation Transform Matrix

The current world space transform is multiplied by this matrix, and the result becomes the new current world space transform. The **Flags** field determines the order of multiplication.

See section 2.3.9 for the specification of additional transform record types.

2.3.9.4 EmfPlusScaleWorldTransform Record

The EmfPlusScaleWorldTransform record performs a scaling on the current world space transform.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type																Flags															
Size																															
DataSize																															
Sx																															
Sy																															

Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusScaleWorldTransform from the RecordType enumeration. The value MUST be 0x402E.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	A	X	X	X	X	X	X	X	X	X	X	X	X	X																

X (1 bit): Reserved and MUST be ignored.

A (1 bit): If set, the transform matrix is post-multiplied. If clear, it is pre-multiplied.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000014.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000008.

Sx (4 bytes): A 32-bit floating-point value that defines the horizontal scale factor. The scaling is performed by constructing a new transform matrix from the **Sx** and **Sy** field values, as shown in the following table.

Sx	0	0
0	Sy	0

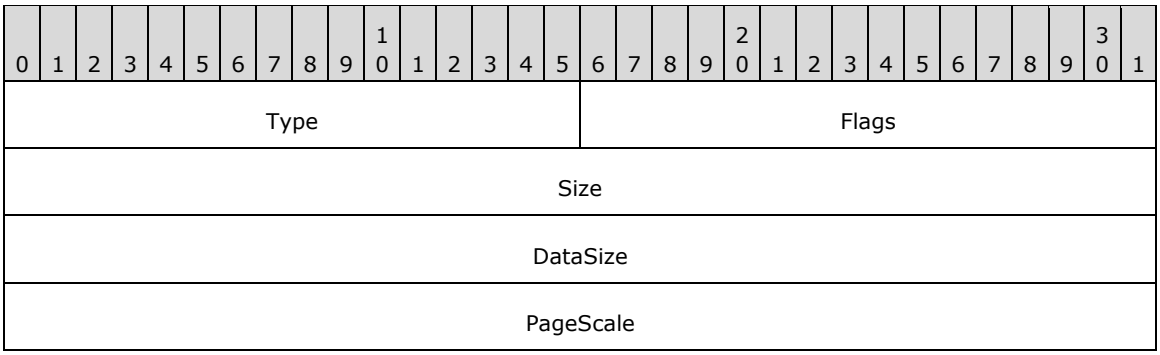
Figure 4: Scale Transform Matrix

Sy (4 bytes): A 32-bit floating-point value that defines the vertical scale factor.

See section 2.3.9 for the specification of additional transform record types.

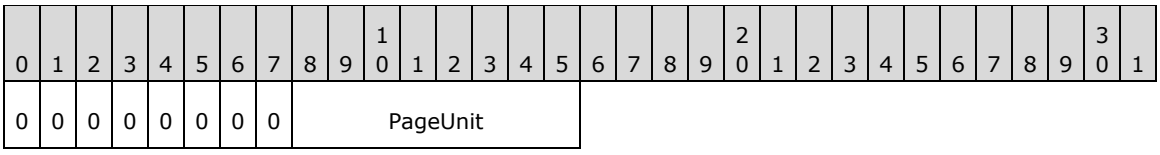
2.3.9.5 EmfPlusSetPageTransform Record

The EmfPlusSetPageTransform record specifies scaling factors and units for converting page space coordinates to device space coordinates.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetPageTransform from the RecordType enumeration. The value MUST be 0x4030.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



PageUnit (1 byte): The unit of measure for page space coordinates, from the UnitType enumeration. This value SHOULD NOT be **UnitTypeDisplay** or **UnitTypeWorld**.<32>

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000010.

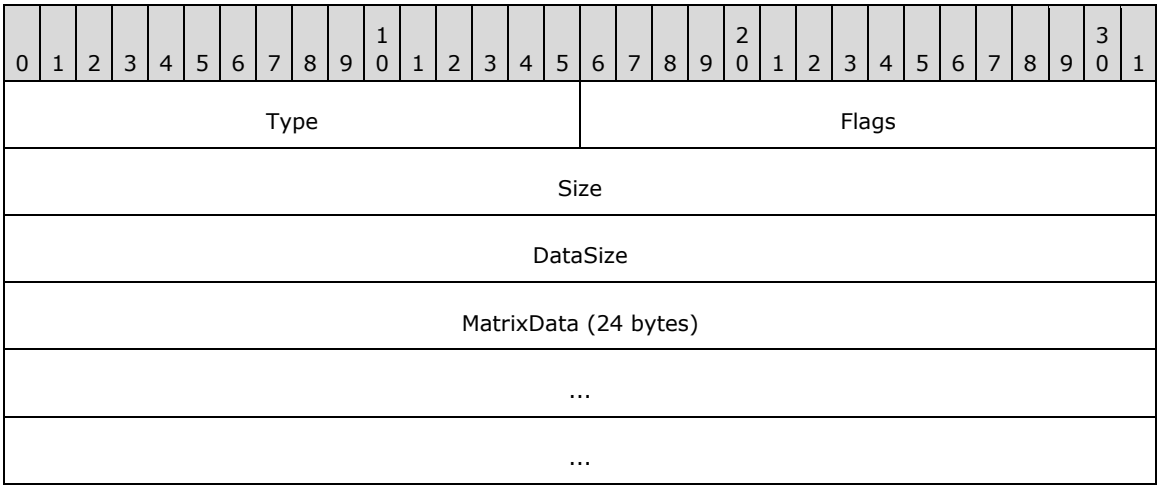
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000004.

PageScale (4 bytes): A 32-bit floating-point value that specifies the scale factor for converting page space coordinates to device space coordinates.

See section 2.3.9 for the specification of additional transform record types.

2.3.9.6 EmfPlusSetWorldTransform Record

The EmfPlusSetWorldTransform record sets the world transform according to the values in a specified transform matrix.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusSetWorldTransform from the RecordType enumeration. The value MUST be 0x402A.

Flags (2 bytes): A 16-bit unsigned integer that is not used. This field SHOULD be set to zero and MUST be ignored upon receipt.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000024.

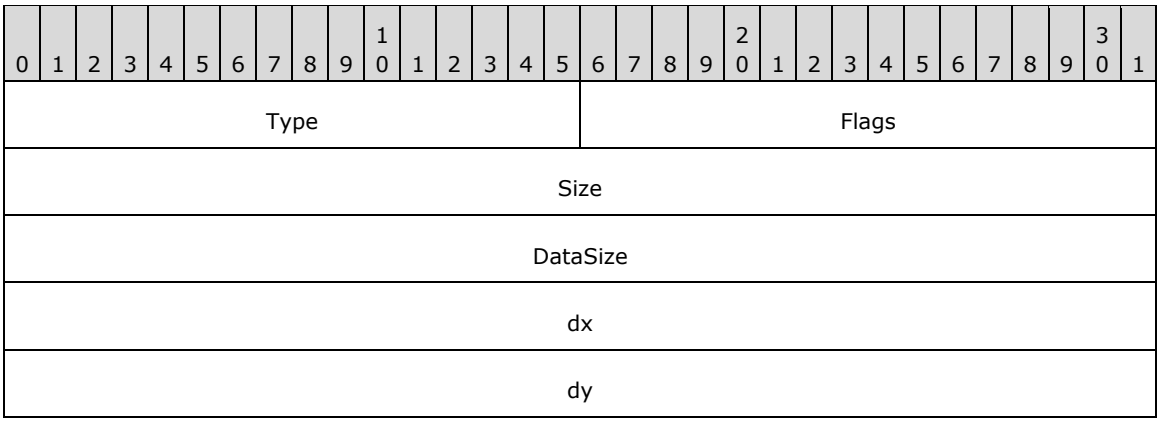
DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000018.

MatrixData (24 bytes): An EmfPlusTransformMatrix object that defines the new current world transform.

See section 2.3.9 for the specification of additional transform record types.

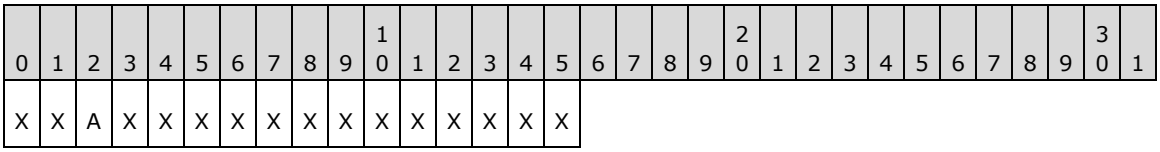
2.3.9.7 EmfPlusTranslateWorldTransform Record

The EmfPlusTranslateWorldTransform record performs a translation on the current world space transform.



Type (2 bytes): A 16-bit unsigned integer that identifies this record type as EmfPlusTranslateWorldTransform from the RecordType enumeration. The value MUST be 0x402D.

Flags (2 bytes): A 16-bit unsigned integer that provides information about how the operation is to be performed, and about the structure of the record.



X (1 bit): Reserved and MUST be ignored.

A (1 bit): If set, the transform matrix is post-multiplied. If clear, it is pre-multiplied.

Size (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes in the entire record, including the 12-byte record header and record-specific data. For this record type, the value MUST be 0x00000014.

DataSize (4 bytes): A 32-bit unsigned integer that specifies the 32-bit-aligned number of bytes of record-specific data that follows. For this record type, the value MUST be 0x00000008.

dx (4 bytes): A 32-bit floating-point value that defines the horizontal distance. The translation is performed by constructing a new world transform matrix from the **dx** and **dy** fields.

dy (4 bytes): A 32-bit floating-point value that defines the vertical distance value.

See section 2.3.9 for the specification of additional transform record types.

3 Structure Examples

The following sections present examples that illustrate the function of EMF+ Extensions structures.

3.1 Metafile Design

3.1.1 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 was 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 needs to 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, machines that are big-endian based will have to perform this necessary compensation.

3.1.2 Managing Graphics Objects

This section describes a conceptual model for Managing Graphics Objects that an EMF+implementation can maintain during metafile playback. The described organization is provided to facilitate the explanation of the file format. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that defined in this specification.

The EMF+ EmfPlusObject record creates graphics objects of the types defined in the GraphicsObjectType enumeration, and the objects themselves are specified in Graphics Objects. Once created, graphics objects can be reused any number of times in graphics operations.

An implementation is responsible for keeping track of graphics objects during metafile playback. A model for managing graphics objects is described below. This model minimizes the space needed by the graphics object table during playback.

1. Create a hash table for graphics objects used in metafile playback.
2. Grow the hash table as objects are created by the EmfPlusObject record; each new entry in the table receives an object and its index.
3. When a metafile record refers to an object, it specifies the object's index. Use this index as a key into the hash table to retrieve the object.
4. When a new object is created that has the same index as an existing object, replace the hash table entry of the existing object with one for the new object.

3.2 EMF+ Metafile Example

This section provides an example of a metafile, which when processed renders the following image:

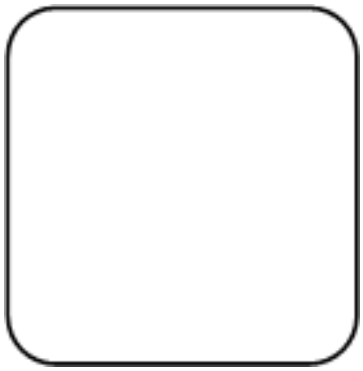


Figure 5: EMF+ Metafile Example

The contents of this metafile example are shown below in hexadecimal bytes. The far-left column is the byte offset from the start of the metafile; the far-right characters are the interpretation of the byte values in the Latin-1 **ANSI character set** [ISO/IEC-8859-1]. The sections that follow describe the metafile records that convey this series of bytes.

```
00000000: 01 00 00 00 6C 00 00 00 FF FF FF FF FF FF FF FF  ...l...ÿÿÿÿÿÿÿÿ
00000010: 64 00 00 00 6B 00 00 00 00 00 00 00 00 00 00 00 d...k...........
00000020: F0 07 00 00 77 08 00 00 20 45 4D 46 00 00 01 00 ð..w .. EMF.. .
00000030: 5C 0A 00 00 4C 00 00 00 02 00 00 00 00 00 00 00 \...L... .......
00000040: 00 00 00 00 00 00 00 00 40 06 00 00 B0 04 00 00 ........@ ..° ..
00000050: 40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00 @ ..ð...........
00000060: 00 00 00 00 00 E2 04 00 80 A9 03 00 46 00 00 00 .....â .€© .F...
00000070: 2C 00 00 00 20 00 00 00 45 4D 46 2B 01 40 01 00 ,... ...EMF+ @ .
00000080: 1C 00 00 00 10 00 00 00 02 10 C0 DB 01 00 00 00  ... ...  ÀÛ ...
00000090: 66 00 00 00 6C 00 00 00 21 00 00 00 08 00 00 00 f...l...!... ...
000000A0: 62 00 00 00 0C 00 00 00 01 00 00 00 21 00 00 00 b....... ...!...
000000B0: 08 00 00 00 62 00 00 00 0C 00 00 00 01 00 00 00  ...b....... ...
000000C0: 21 00 00 00 08 00 00 00 21 00 00 00 08 00 00 00 !... ...!... ...
000000D0: 73 00 00 00 0C 00 00 00 00 00 00 00 1C 00 00 00 s........... ...
000000E0: 08 00 00 00 25 00 00 00 0C 00 00 00 00 00 00 80  ...%..........€
000000F0: 25 00 00 00 0C 00 00 00 07 00 00 80 25 00 00 00 %.........€%...
00000100: 0C 00 00 00 0E 00 00 80 30 00 00 00 0C 00 00 00 ........€0.......
00000110: 0F 00 00 80 19 00 00 00 0C 00 00 00 FF FF FF 00  ..€ .......ÿÿÿ.
```

```
00000120: 18 00 00 00 0C 00 00 00 00 00 00 00 12 00 00 00  ..........  ...
00000130: 0C 00 00 00 02 00 00 00 13 00 00 00 0C 00 00 00 .... ... .......
00000140: 01 00 00 00 14 00 00 00 0C 00 00 00 0D 00 00 00  ... ..........
00000150: 15 00 00 00 0C 00 00 00 01 00 00 00 16 00 00 00  ....... ... ...
00000160: 0C 00 00 00 00 00 00 00 0D 00 00 00 10 00 00 00 ............ ...
00000170: 00 00 00 00 00 00 00 00 3A 00 00 00 0C 00 00 00 ........:.......
00000180: 0A 00 00 00 1B 00 00 00 10 00 00 00 00 00 00 00 .... ... .......
00000190: 00 00 00 00 23 00 00 00 20 00 00 00 87 C3 81 3F ....#... ...‡Ã□?
000001A0: 00 00 00 00 00 00 00 00 7A BD 80 3F 00 00 00 80 ........z½€?...€
000001B0: 00 00 00 80 24 00 00 00 24 00 00 00 87 C3 81 3F ...€$...$...‡Ã□?
000001C0: 00 00 00 00 00 00 00 00 7A BD 80 3F 00 00 00 80 ........z½€?...€
000001D0: 00 00 00 80 04 00 00 00 73 00 00 00 0C 00 00 00 ...€ ...s.......
000001E0: 00 00 00 00 0D 00 00 00 10 00 00 00 00 00 00 00 ........ .......
000001F0: 00 00 00 00 52 00 00 00 70 01 00 00 01 00 00 00 ....R...p .. ...
00000200: 10 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00  ..............
00000210: BC 02 00 00 00 00 00 00 07 02 02 22 53 00 79 00 ¼ ......  "S.y.
00000220: 73 00 74 00 65 00 6D 00 00 00 00 00 00 00 00 00 s.t.e.m.........
00000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000260: B1 E6 31 7C BC FE 12 00 21 00 00 00 00 00 13 00 ±æ1|¼þ .!..... .
00000270: 00 00 2E 00 54 E1 12 00 BC FE 12 00 14 E1 12 00 ....Tá .¼þ . á .
00000280: 00 00 00 00 F4 F4 12 00 34 B8 F9 77 68 26 F4 77 ....ôô .4¸ùwh&ôw
00000290: FF FF FF FF AD 97 F7 77 7F 99 F7 77 70 E1 12 00 ÿÿÿÿ-—÷w™÷wpá .
000002A0: 21 00 00 00 01 00 00 00 FC 8B CE 02 38 A6 17 00 !... ...ü‹Î 8¦ .
000002B0: 7F 99 F7 77 B0 DA 4F 00 63 DE 5D 00 3E 08 5A 00 ™÷w°ÚO.cÞ].> Z.
000002C0: 68 99 5A 00 F5 82 54 00 1A 91 54 00 8F 9B C7 77 h™Z.õ‚T. 'T.□›Çw
000002D0: BD 93 54 00 6F 9D 54 00 94 E5 54 00 00 00 54 00 ½"T.o□T."åT...T.
000002E0: AF 58 56 00 6B 50 56 00 B4 3A 55 00 05 39 55 00 ¯XV.kPV.':U. 9U.
000002F0: 09 35 55 00 64 C4 4F 00 00 82 41 00 FF 44 41 00 .5U.dÄO..,A.ÿDA.
00000300: 12 4E 41 00 E1 4B 41 00 1D 1E 31 7C 4B 16 31 7C  NA.áKA.  1|K 1|
00000310: DA EF 30 7C 49 F4 30 7C EA A3 37 7C 00 00 D5 77 Úï0|Iô0|ê£7|..Õw
00000320: A5 DC D5 77 46 46 D3 77 D7 96 D3 77 97 ED 31 7C ¥ÜÕwFFÓw×-Ów—í1|
00000330: B1 E6 31 7C 00 00 2E 01 10 E2 12 00 68 8B CE 02 ±æ1|...  â .h‹Î
00000340: 08 ED F8 77 68 8B CE 02 00 00 2E 01 01 00 00 00  íøwh‹Î ...  ...
00000350: 00 00 2E 01 C4 04 F9 77 27 05 F9 77 64 76 00 08 ... Ä ùw' ùwdv.
00000360: 00 00 00 00 25 00 00 00 0C 00 00 00 01 00 00 00 ....%....... ...
00000370: 25 00 00 00 0C 00 00 00 0E 00 00 80 28 00 00 00 %.........€(...
00000380: 0C 00 00 00 01 00 00 00 46 00 00 00 88 02 00 00 .... ...F...ˆ ..
00000390: 7C 02 00 00 45 4D 46 2B 1E 40 0B 00 0C 00 00 00 | ..EMF+ @......
000003A0: 00 00 00 00 24 40 02 00 0C 00 00 00 00 00 00 00 ....$@ .........
000003B0: 21 40 07 00 0C 00 00 00 00 00 00 00 22 40 03 00 !@.........."@ .
000003C0: 0C 00 00 00 00 00 00 00 1F 40 05 00 0C 00 00 00 .........¬@ .....
000003D0: 00 00 00 00 2C 40 00 00 24 00 00 00 18 00 00 00 ....,@..$... ...
000003E0: 00 00 80 3F 00 00 00 00 00 00 00 00 00 00 80 3F ..€?..........€?
000003F0: 00 00 00 80 00 00 00 80 25 40 00 00 10 00 00 00 ...€...€%@.. ...
00000400: 04 00 00 00 00 00 00 00 2C 40 00 00 24 00 00 00  ........,@..$...
00000410: 18 00 00 00 00 00 80 3F 00 00 00 00 00 00 00 00  ......€?........
00000420: 00 00 80 3F 00 00 00 00 00 00 00 00 2A 40 00 00 ..€?........*@..
00000430: 24 00 00 00 18 00 00 00 00 00 80 3F 00 00 00 00 $... ......€?....
00000440: 00 00 00 00 00 00 80 3F 00 00 00 80 00 00 00 80 .......€?...€...€
00000450: 28 40 00 00 10 00 00 00 04 00 00 00 01 00 00 00 (@.. ... ... ...
00000460: 1E 40 0B 00 0C 00 00 00 00 00 00 00 24 40 02 00  @..........$@ .
00000470: 0C 00 00 00 00 00 00 00 21 40 07 00 0C 00 00 00 ........!@......
00000480: 00 00 00 00 22 40 03 00 0C 00 00 00 00 00 00 00 ...."@ .........
00000490: 1F 40 05 00 0C 00 00 00 00 00 00 00 30 40 02 00 ¬@ .........0@ .
000004A0: 10 00 00 00 04 00 00 00 00 00 80 3F 2A 40 00 00  ... ......€?*@..
000004B0: 24 00 00 00 18 00 00 00 00 00 80 3F 00 00 00 00 $... ......€?....
000004C0: 00 00 00 00 00 00 80 3F 00 00 00 80 00 00 00 80 .......€?...€...€
000004D0: 2A 40 00 00 24 00 00 00 18 00 00 00 00 00 80 3F *@..$... ......€?
000004E0: 00 00 00 00 00 00 00 00 00 00 80 3F 00 00 00 80 ...........€?...€
000004F0: 00 00 00 80 2A 40 00 00 24 00 00 00 18 00 00 00 ...€*@..$... ...
00000500: 0D 74 DA 3A 00 00 00 00 00 00 00 00 0D 74 DA 3A .tÚ:.........tÚ:
00000510: 00 00 00 80 00 00 00 80 2A 40 00 00 24 00 00 00 ...€...€*@..$...
00000520: 18 00 00 00 92 5F 2C 3E 00 00 00 00 00 00 00 00  ...' ,>........
00000530: EB 51 38 3E 00 00 00 80 00 00 00 80 08 40 00 03 ëQ8>...€...€ @.
00000540: C4 00 00 00 B8 00 00 00 02 10 C0 DB 13 00 00 00 Ä...¸...  ÀÛ ...
00000550: 00 00 00 00 FC 7F F5 43 CF FF FF BF 9D 8E 08 44 ....üõCÏÿÿ¿□Ž D
00000560: 1E 01 00 C0 FE BF 13 44 EB 15 2B 42 FF BF 13 44   .Àþ¿ Dë +Bÿ¿ D
```

```
00000570: FC FF C3 42 FF BF 13 44 FE FF C3 42 FF BF 13 44 üÿÃBÿ¿ DþÿÃBÿ¿ D
00000580: 01 00 C4 42 FF BF 13 44 03 00 C4 42 FF BF 13 44  .ÄBÿ¿ D .ÄBÿ¿ D
00000590: FF FF F5 43 FF BF 13 44 9F CE 08 44 9F 8E 08 44 ÿÿõCÿ¿ DŸÎ DŸŽ D
000005A0: FF FF 13 44 00 80 F5 43 FF FF 13 44 05 00 C2 42 ÿÿ D.€õCÿÿ D .ÂB
000005B0: FF FF 13 44 16 16 27 42 00 00 14 44 72 FF 3F C0 ÿÿ D  'B.. Drÿ?À
000005C0: 9F CE 08 44 E8 FF 3F C0 01 00 F6 43 10 00 40 C0 ŸÎ Dèÿ?À .öC .@À
000005D0: 04 00 C4 42 64 00 40 C0 17 16 2B 42 FA 15 27 42  .ÄBd.@À  +Bú 'B
000005E0: E8 FE FF BF F6 FF C1 42 26 00 00 C0 00 03 03 03 èþÿ¿öÿÁB&..À.
000005F0: 03 03 03 01 03 03 03 01 03 03 03 01 03 03 83 BF              ƒ¿
00000600: 14 40 00 80 10 00 00 00 04 00 00 00 FF FF FF 00  @.€ ... ...ÿÿÿ.
00000610: 46 00 00 00 50 00 00 00 44 00 00 00 45 4D 46 2B F...P...D...EMF+
00000620: 08 40 01 02 30 00 00 00 24 00 00 00 02 10 C0 DB  @  0...$...  ÀÛ
00000630: 00 00 00 00 80 00 00 00 02 00 00 00 00 00 80 3F ....€... .....€?
00000640: 00 00 00 00 02 10 C0 DB 00 00 00 00 00 00 00 FF ....  ÀÛ.......ÿ
00000650: 15 40 00 00 10 00 00 00 04 00 00 00 01 00 00 00  @.. ... ... ...
00000660: 22 00 00 00 0C 00 00 00 FF FF FF FF 22 00 00 00 ".......ÿÿÿÿ"...
00000670: 0C 00 00 00 FF FF FF FF 25 00 00 00 0C 00 00 00 ....ÿÿÿÿ%.......
00000680: 0D 00 00 80 22 00 00 00 0C 00 00 00 FF FF FF FF ...€".......ÿÿÿÿ
00000690: 21 00 00 00 08 00 00 00 73 00 00 00 0C 00 00 00 !... ...s.......
000006A0: 00 00 00 00 1C 00 00 00 08 00 00 00 25 00 00 00 .... ... ...%...
000006B0: 0C 00 00 00 00 00 00 80 25 00 00 00 0C 00 00 00 .......€%.......
000006C0: 07 00 00 80 25 00 00 00 0C 00 00 00 0E 00 00 80 ..€%.........€
000006D0: 30 00 00 00 0C 00 00 00 0F 00 00 80 19 00 00 00 0....... ..€ ...
000006E0: 0C 00 00 00 FF FF FF 00 18 00 00 00 0C 00 00 00 ....ÿÿÿ. .......
000006F0: 00 00 00 00 12 00 00 00 0C 00 00 00 02 00 00 00 .... ....... ...
00000700: 13 00 00 00 0C 00 00 00 01 00 00 00 14 00 00 00  ....... ... ...
00000710: 0C 00 00 00 0D 00 00 00 15 00 00 00 0C 00 00 00 ........ .......
00000720: 01 00 00 00 16 00 00 00 0C 00 00 00 00 00 00 00  ... ............
00000730: 0D 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 .... ............
00000740: 3A 00 00 00 0C 00 00 00 0A 00 00 00 1B 00 00 00 :............ ...
00000750: 10 00 00 00 00 00 00 00 00 00 00 00 23 00 00 00  ............#...
00000760: 20 00 00 00 87 C3 81 3F 00 00 00 00 00 00 00 00  ...‡Ã?........
00000770: 7A BD 80 3F 00 00 00 80 00 00 00 80 24 00 00 00 z½€?...€...€$...
00000780: 24 00 00 00 87 C3 81 3F 00 00 00 00 00 00 00 00 $...‡Ã?........
00000790: 7A BD 80 3F 00 00 00 80 00 00 00 80 04 00 00 00 z½€?...€...€ ...
000007A0: 73 00 00 00 0C 00 00 00 00 00 00 00 46 00 00 00 s...........F...
000007B0: 60 00 00 00 52 00 00 00 47 44 49 43 02 00 00 00 `...R...GDIC ...
000007C0: 00 00 00 00 00 00 00 00 66 00 00 00 6C 00 00 00 ........f...l...
000007D0: 1B 00 00 00 41 00 70 00 70 00 4E 00 61 00 6D 00  ...A.p.p.N.a.m.
000007E0: 65 00 00 00 49 00 6D 00 61 00 67 00 65 00 20 00 e...I.m.a.g.e. .
000007F0: 44 00 65 00 73 00 63 00 72 00 69 00 70 00 74 00 D.e.s.c.r.i.p.t.
00000800: 69 00 6F 00 6E 00 00 00 00 00 00 00 23 00 00 00 i.o.n.......#...
00000810: 20 00 00 00 FF FF 7F 3F 00 00 00 00 00 00 00 00  ...ÿÿ?........
00000820: FD FF 7F 3F 00 00 00 00 00 00 00 00 24 00 00 00 ýÿ?........$...
00000830: 24 00 00 00 FF FF 7F 3F 00 00 00 00 00 00 00 00 $...ÿÿ?........
00000840: FD FF 7F 3F 00 00 00 00 00 00 00 00 04 00 00 00 ýÿ?........ ...
00000850: 23 00 00 00 20 00 00 00 FF FF 7F 3F 00 00 00 00 #... ...ÿÿ?....
00000860: 00 00 00 00 FD FF 7F 3F 00 00 00 00 00 00 00 00 ....ýÿ?........
00000870: 24 00 00 00 24 00 00 00 FF FF 7F 3F 00 00 00 00 $...$...ÿÿ?....
00000880: 00 00 00 00 FD FF 7F 3F 00 00 00 00 00 00 00 00 ....ýÿ?........
00000890: 04 00 00 00 23 00 00 00 20 00 00 00 0D 74 DA 3A  ...#... ....tÚ:
000008A0: 00 00 00 00 00 00 00 00 0C 74 DA 3A 00 00 00 00 .........tÚ:....
000008B0: 00 00 00 00 24 00 00 00 24 00 00 00 0D 74 DA 3A ....$...$....tÚ:
000008C0: 00 00 00 00 00 00 00 00 0C 74 DA 3A 00 00 00 00 .........tÚ:....
000008D0: 00 00 00 00 04 00 00 00 23 00 00 00 20 00 00 00 .... ...#... ...
000008E0: 92 5F 2C 3E 00 00 00 00 00 00 00 00 E9 51 38 3E '_,>........éQ8>
000008F0: 00 00 00 00 00 00 00 00 24 00 00 00 24 00 00 00 ........$...$...
00000900: 92 5F 2C 3E 00 00 00 00 00 00 00 00 E9 51 38 3E '_,>........éQ8>
00000910: 00 00 00 00 00 00 00 00 04 00 00 00 25 00 00 00 ........ ...%...
00000920: 0C 00 00 00 05 00 00 80 2C 00 00 00 20 00 00 00 .... ..€,... ...
00000930: FD FF FF FF FE FF FF FF 4F 02 00 00 50 02 00 00 ýÿÿÿþÿÿÿO ..P ..
00000940: C8 00 00 00 C8 00 00 00 46 00 00 00 14 00 00 00 È...È...F... ...
00000950: 08 00 00 00 47 44 49 43 03 00 00 00 22 00 00 00  ...GDIC ..."...
00000960: 0C 00 00 00 FF FF FF FF 46 00 00 00 54 00 00 00 ....ÿÿÿÿF...T...
00000970: 48 00 00 00 45 4D 46 2B 29 40 00 00 10 00 00 00 H...EMF+)@.. ...
00000980: 04 00 00 00 01 00 00 00 26 40 00 00 10 00 00 00  ... ...&@.. ...
00000990: 04 00 00 00 00 00 00 00 2A 40 00 00 24 00 00 00  ........*@..$...
000009A0: 18 00 00 00 00 00 00 80 3F 00 00 00 00 00 00 00  ......€?........
000009B0: 00 00 80 3F 00 00 00 00 00 00 00 00 4C 00 00 00 ..€?........L...
```

```
000009C0: 64 00 00 00 FF FF FF FF FF FF FF FF 64 00 00 00 d...ÿÿÿÿÿÿÿÿd...
000009D0: 6B 00 00 00 FF FF FF FF FF FF FF FF 66 00 00 00 k...ÿÿÿÿÿÿÿÿf...
000009E0: 6D 00 00 00 29 00 AA 00 00 00 00 00 00 00 00 00 m...).ª.........
000009F0: 00 00 80 3F 00 00 00 00 00 00 00 00 00 00 80 3F ..€?..........€?
00000A00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000A10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000A20: 22 00 00 00 0C 00 00 00 FF FF FF FF 46 00 00 00 ".......ÿÿÿÿF...
00000A30: 1C 00 00 00 10 00 00 00 45 4D 46 2B 02 40 00 00  ... ...EMF+ @..
00000A40: 0C 00 00 00 00 00 00 00 0E 00 00 00 14 00 00 00 ............ ...
00000A50: 00 00 00 00 10 00 00 00 14 00 00 00
```

The sections that follow provide definitions of the header and records that correspond to this metafile. These sections describe each record as it appears in the hexadecimal representation of the example, and the records have been interpreted for the convenience of the reader; however, to reflect the exact order of records in the metafile, some record types appear more than once, and, in some cases, the repeated instances can be identical.

3.2.1 EMR_HEADER Example

This section provides an example of the EMF EMR_HEADER record.

```
00000000: 01 00 00 00 6C 00 00 00 FF FF FF FF FF FF FF FF
00000010: 64 00 00 00 6B 00 00 00 00 00 00 00 00 00 00 00
00000020: F0 07 00 00 77 08 00 00 20 45 4D 46 00 00 01 00
00000030: 5C 0A 00 00 4C 00 00 00 02 00 00 00 00 00 00 00
00000040: 00 00 00 00 00 00 00 00 40 06 00 00 B0 04 00 00
00000050: 40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 E2 04 00 80 A9 03 00
```

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
Type (0x00000001)																															
Size (0x0000006C)																															
Bounds (0xFFFFFFFF)																															
... (0xFFFFFFFF)																															
... (0x00000064)																															
... (0x0000006B)																															
Frame (0x00000000)																															
... (0x00000000)																															
... (0x000007F0)																															
... (0x00000877)																															

Figure 6: EMF EMR_HEADER Record Example, Part 1

Type: 0x00000001 identifies the record type as EMR_HEADER.

Size: 0x0000006C specifies the record size in bytes, 108.

Bounds: (0xFFFFFFFF, 0xFFFFFFFF, 0x00000064, 0x0000006B) specifies the rectangular inclusive-inclusive bounds in device units of the smallest rectangle that can be drawn around the image stored in the metafile.

Frame: (128-bits) (0x00000000, 0x00000000, 0x000007F0, 0x00000877) specifies the rectangular inclusive-inclusive dimensions, in .01 millimeter units, of a rectangle that surrounds the image stored in the metafile.

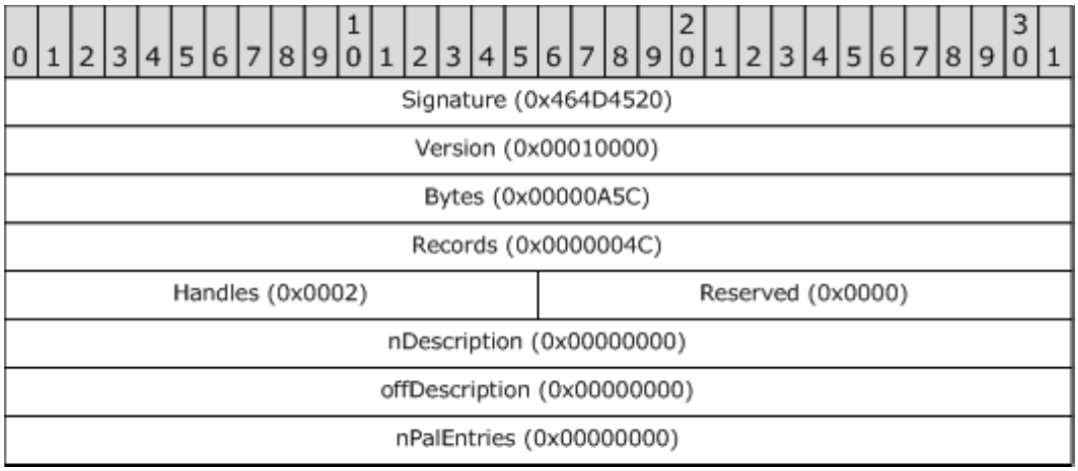


Figure 7: EMF EMR_HEADER Record Example, Part 2

Signature: 0x464D4520 specifies the record signature, which consists of the **ASCII** string " EMF".

Version: 0x00010000 specifies EMF metafile interoperability.

Bytes: 0x00000A5C specifies the size of the metafile in bytes.

Records: 0x0000004C specifies the number of records in the metafile

Handles: 0x0002 specifies the number of indexes that will need to be defined during the processing of the metafile. These indexes correspond to graphics objects that are used in drawing commands. Index zero is reserved for references to the metafile itself.

Reserved: 0x0000 is not used.

nDescription: 0x00000000 specifies the number of characters in the array that contains the description of the metafile's contents. Zero indicates there is no description string.

offDescription: 0x00000000 specifies the offset from the beginning of this record to the array that contains the description of the metafile's contents.

nPalEntries: 0x00000000 specifies the number of entries in the metafile palette. The location of the palette is specified in the EMF end-of-file record, EMR_EOF.

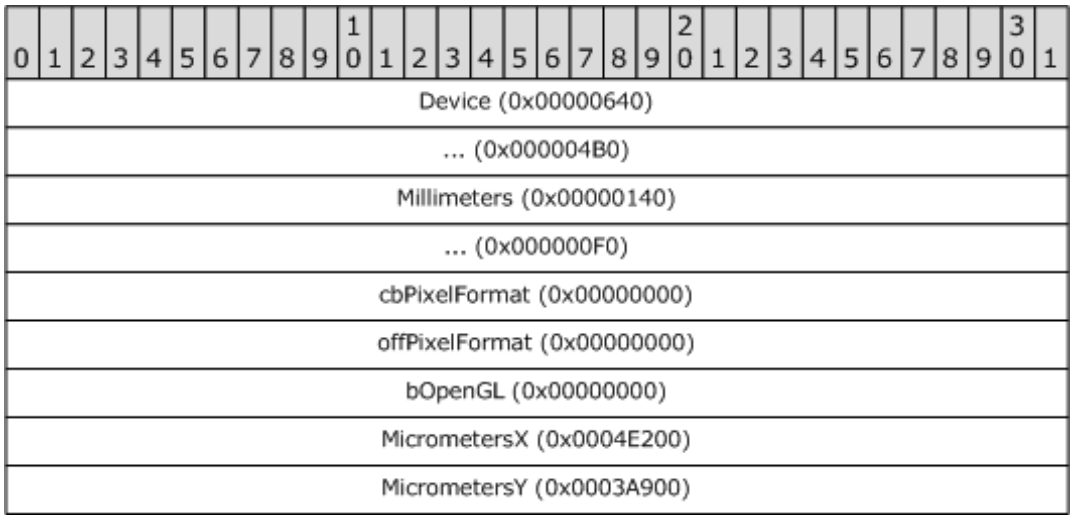


Figure 8: EMF EMR_HEADER Record Example, Part 3

Device: (0x00000640, 0x000004B0) specifies the size of the reference device in pixels.

Millimeters: (0x00000140, 0x000000F0) specifies the size of the reference device in millimeters.

cbPixelFormat: 0x00000000 specifies the size of the EMF PixelFormatDescriptor object. This value indicates that no pixel format is defined.

offPixelFormat: 0x00000000 specifies the offset to the PixelFormatDescriptor in the metafile. In this case, no pixel format structure is present.

bOpenGL: 0x00000000 specifies that no EMF OpenGL records [OPENGL] are present in the metafile.

MicrometersX: 0x0004E200 specifies the horizontal size of the reference device in micrometers.

MicrometersY: 0x0003A900 specifies the vertical size of the reference device in micrometers.

3.2.2 EMR_COMMENT_EMFPLUS Example 1

This section provides an example of the EMF EMR_COMMENT_EMFPLUS record.

```
00000060:                                     46 00 00 00
00000070: 2C 00 00 00 20 00 00 00 45 4D 46 2B
```

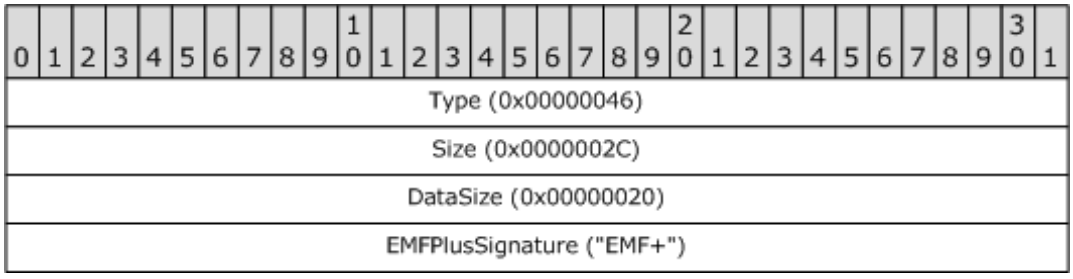


Figure 9: EMF EMR_COMMENT_EMFPLUS Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x0000002C specifies the size of this record in bytes, including embedded EMF+ records.

DataSize: 0x00000020 specifies the size of the EMF+ data in bytes.

EMFPlusSignature: "EMF+" identifies the comment record type as EMR_COMMENT_EMFPLUS.

The embedded EMF+ records are presented in the sections that follow.

3.2.2.1 EmfPlusHeader Example

This section provides an example of the EmfPlusHeader record.

```
00000070:                                       01 40 01 00
00000080: 1C 00 00 00 10 00 00 00 02 10 C0 DB 01 00 00 00
00000090: 66 00 00 00 6C 00 00 00
```

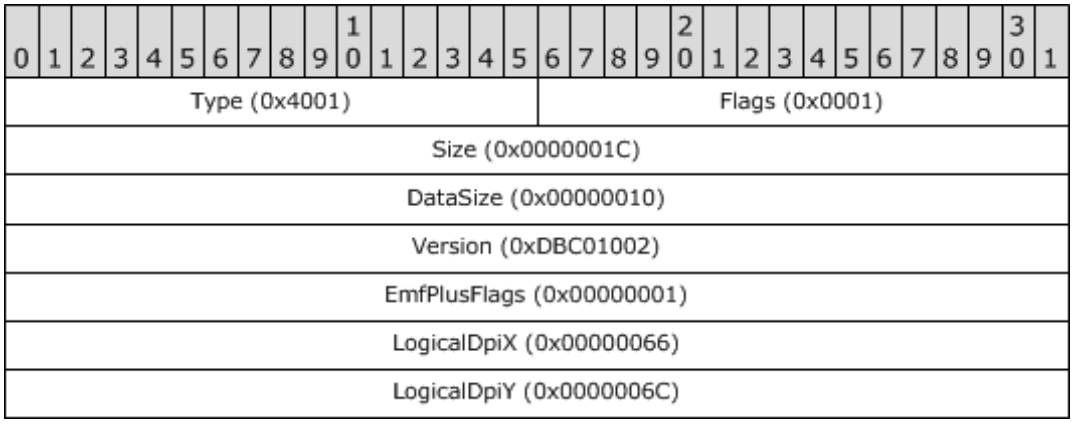


Figure 10: EmfPlusHeader Record Example

Type: 0x4001 identifies this record type as an EmfPlusHeader.

Flags: 0x0001 specifies that the metafile is **EMF+ Dual**.

Size: 0x0000001C specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

DataSize: 0x00000010 specifies the number of bytes of data in the **RecordData** member that follows. This number does not include the size of the invariant part of this record.

Version: 0xDBC01002 specifies the version of operating systems graphics that was used to create the metafile (EmfPlusGraphicsVersion object).

EmfPlusFlags: 0x00000001 specifies the additional information about how this metafile was recorded. If the value is set to 0x00000001, the metafile was recorded with a reference playback device context for a video display.

LogicalDpiX: 0x00000066 specifies the dots per inch (DPI) in the horizontal direction of the drawing surface for which the metafile was created.

LogicalDpiY: 0x0000006C specifies the dots per inch (DPI) in the vertical direction of the drawing surface for which the metafile was created.

3.2.3 EMR_SAVEDC Example 1

This section provides an example of the EMR_SAVEDC record.

```
00000090:                         21 00 00 00 08 00 00 00
```

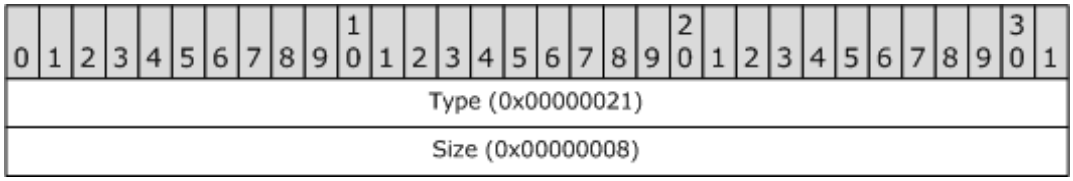


Figure 11: EMF EMR_SAVEDC Record Example

Type: 0x00000021 identifies this EMF record type as EMR_SAVEDC.

Size: 0x00000008 specifies the size of this record in bytes.

3.2.4 EMR_SETICMMODE Example 1

This section provides an example of the EMF EMR_SETICMMODE record.

```
000000A0: 62 00 00 00 0C 00 00 00 01 00 00 00
```

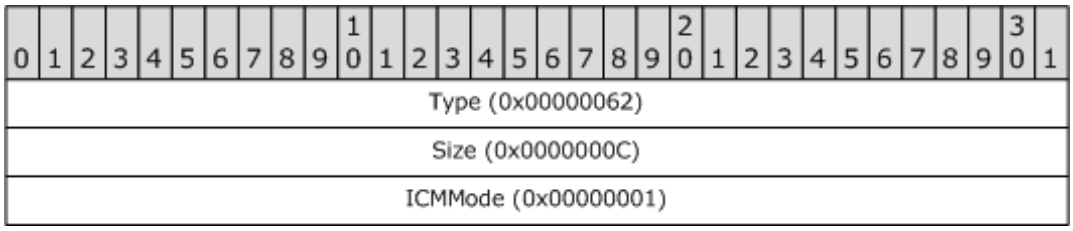


Figure 12: EMF EMR_SETICMMODE Record Example

Type: 0x00000062 identifies the record type as EMR_SETICMMODE.

Size: 0x0000000C specifies the size of this record in bytes.

ICMMode: 0x00000001 specifies turning off **Image Color Management (ICM)**.This value is defined in the EMF ICMMode enumeration.

3.2.5 EMR_SAVEDC Example 2

This section provides an example of the EMR_SAVEDC record.

```
000000A0:                                  21 00 00 00
000000B0: 08 00 00 00
```

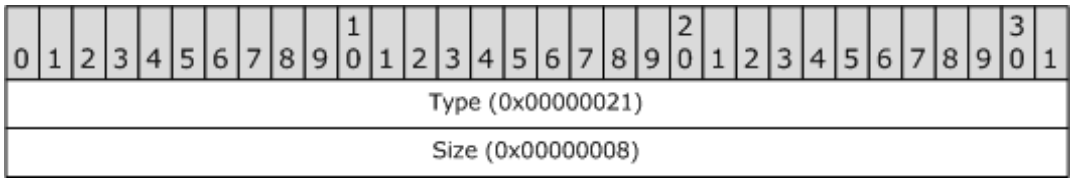


Figure 13: EMF EMR_SAVEDC Record Example

Type: 0x00000021 identifies this EMF record type as EMR_SAVEDC.

Size: 0x00000008 specifies the size of this record in bytes.

3.2.6 EMR_SETICMMODE Example 2

This section provides an example of the EMF EMR_SETICMMODE record.

```
000000B0:             62 00 00 00 0C 00 00 00 01 00 00 00
```

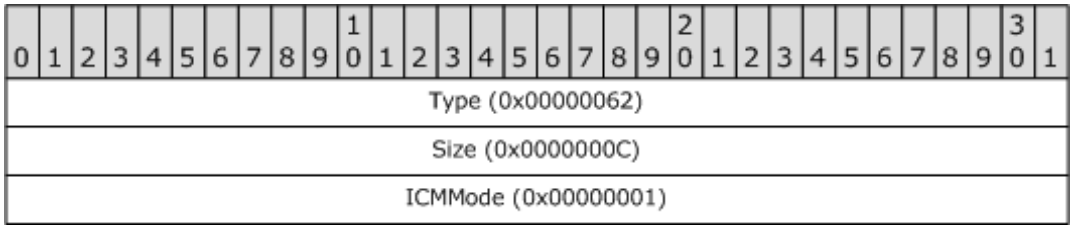


Figure 14: EMF EMR_SETICMMODE Record Example

Type: 0x00000062 identifies the record type as EMR_SETICMMODE.

Size: 0x0000000C specifies the size of this record in bytes.

ICMMode: 0x00000001 specifies turning off Image Color Management (ICM). This value is defined in the EMF ICMMode enumeration.

3.2.7 EMR_SAVEDC Example 3

This section provides an example of the EMR_SAVEDC record.

```
000000C0: 21 00 00 00 08 00 00 00
```

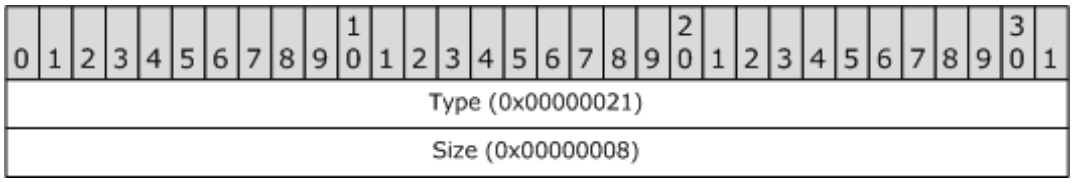


Figure 15: EMF EMR_SAVEDC Record Example

Type: 0x00000021 identifies this EMF record type as EMR_SAVEDC.

Size: 0x00000008 specifies the size of this record in bytes.

3.2.8 EMR_SETLAYOUT Example 1

This section provides an example of the EMF EMR_SETLAYOUT record.

```
000000D0: 73 00 00 00 0C 00 00 00 00 00 00 00
```

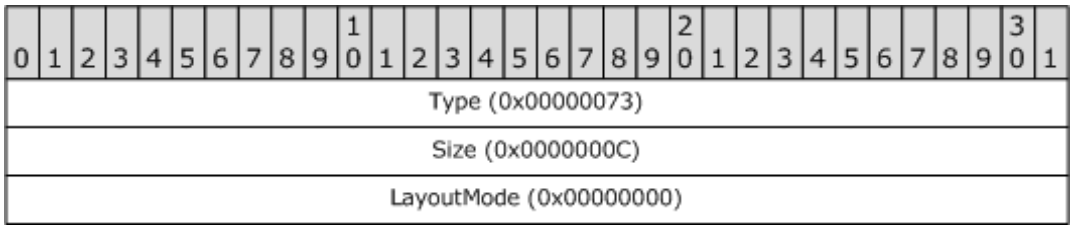


Figure 16: EMF EMR_SETLAYOUT Record Example

Type: 0x00000073 identifies this record type as EMR_SETLAYOUT.

Size: 0x0000000C specifies the size of this record in bytes.

LayoutMode: 0x00000000 specifies left-to-right horizontal layout.

3.2.9 EMR_SETMETARGN Example 1

This section provides an example of the EMR_SETMETARGN record.

```
000000D0:                                     1C 00 00 00
000000E0: 08 00 00 00
```

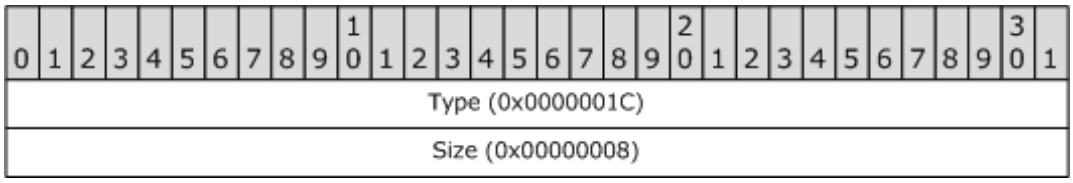


Figure 17: EMF EMR_SETMETARGN Record Example

Type: 0x0000001C identifies this EMF record type as EMR_SETMETARGN.

Size: 0x00000008 specifies the size of this record.

3.2.10 EMR_SELECTOBJECT Example 1

This section provides an example of the EMR_SELECTOBJECT record.

```
000000E0:            25 00 00 00 0C 00 00 00 00 00 00 80
```

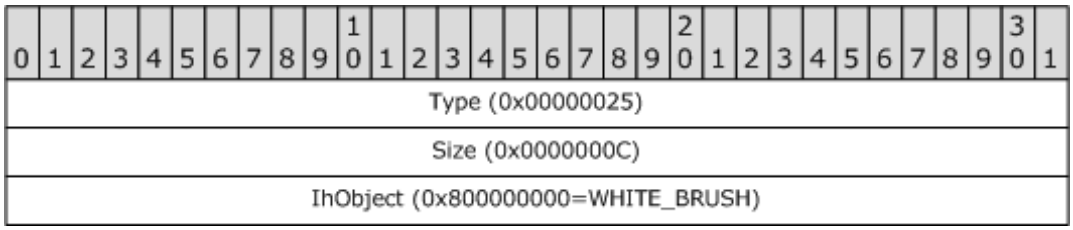


Figure 18: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x80000000 specifies the index of an object in the object table.

3.2.11 EMR_SELECTOBJECT Example 2

This section provides an example of the EMR_SELECTOBJECT record.

```
000000F0: 25 00 00 00 0C 00 00 00 07 00 00 80
```

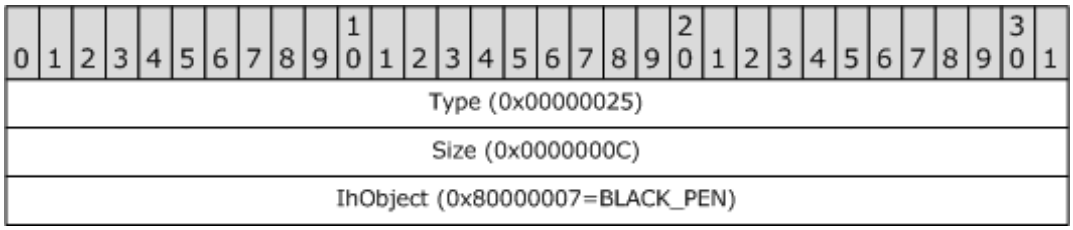


Figure 19: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x80000007 specifies the index of an object in the object table.

3.2.12 EMR_SELECTOBJECT Example 3

This section provides an example of the EMR_SELECTOBJECT record.

```
000000F0:                          25 00 00 00
00000100: 0C 00 00 00 0E 00 00 80
```

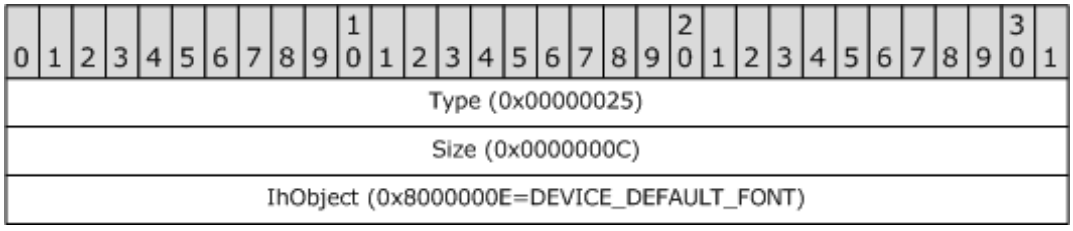


Figure 20: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x8000000E specifies the index of an object in the object table.

3.2.13 EMR_SELECTPALETTE Example 1

This section provides an example of the EMR_SELECTPALETTE record.

```
00000100:               30 00 00 00 0C 00 00 00
00000110: 0F 00 00 80
```

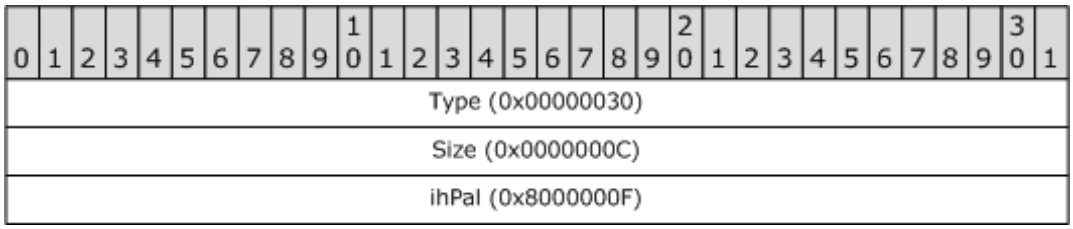


Figure 21: EMF EMR_SELECTPALETTE Record Example

Type: 0x00000030 identifies this EMF record type as EMR_SELECTPALETTE.

Size: 0x0000000C specifies the size of this record in bytes.

ihPal: 0x8000000F specifies the palette index. The palette can be selected in background mode only.

3.2.14 EMR_SETBKCOLOR Example 1

This section provides an example of the EMR_SETBKCOLOR record.

```
00000110:    19 00 00 00 0C 00 00 00 FF FF FF 00
```

Figure 22: EMF EMR_SETBKCOLOR Record Example

Type: 0x00000019 identifies this EMF record type as EMR_SETBKCOLOR.

Size: 0x0000000C specifies the size of this record in bytes.

Color: 0x00FFFFFF specifies the background color value with the WMF ColorRef object [MS-WMF].

3.2.15 EMR_SETTEXTCOLOR Example 1

This section provides an example of the EMR_SETTEXTCOLOR record.

```
00000120: 18 00 00 00 0C 00 00 00 00 00 00 00
```

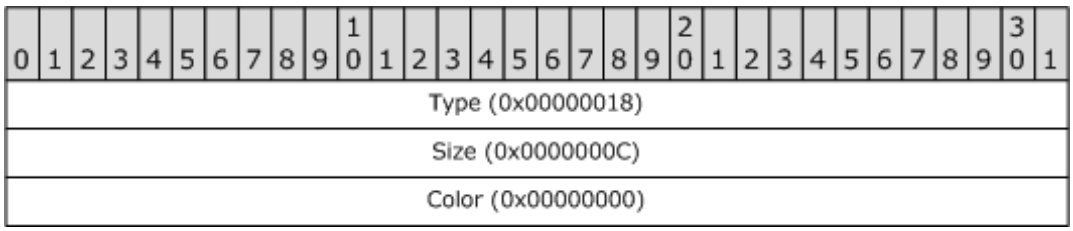


Figure 23: EMF EMR_SETTEXTCOLOR Record Example

Type: 0x00000018 identifies this EMF record type as EMR_SETTEXTCOLOR.

Size: 0x0000000C specifies the size of this record in bytes.

Color: 0x00000000 specifies the text color value in the form specified using the WMF ColorRef object [MS-WMF].

3.2.16 EMR_SETBKMODE Example 1

This section provides an example of the EMF EMR_SETBKMODE record.

```
00000120:                          12 00 00 00
00000130: 0C 00 00 00 02 00 00 00
```

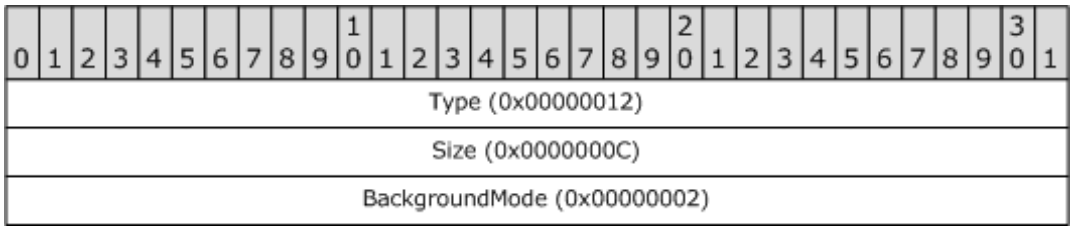


Figure 24: EMF EMR_SETBKMODE Record Example

Type: 0x00000012 identifies the record type as EMR_SETBKMODE.

Size: 0x0000000C specifies the size of this record in bytes.

BackgroundMode: 0x00000002 specifies **OPAQUE** background mode. This value is defined in the EMF BackgroundMode enumeration.

3.2.17 EMR_SETPOLYFILLMODE Example 1

This section provides an example of the EMF EMR_SETPOLYFILLMODE record.

```
00000130:                13 00 00 00 0C 00 00 00
00000140: 01 00 00 00
```

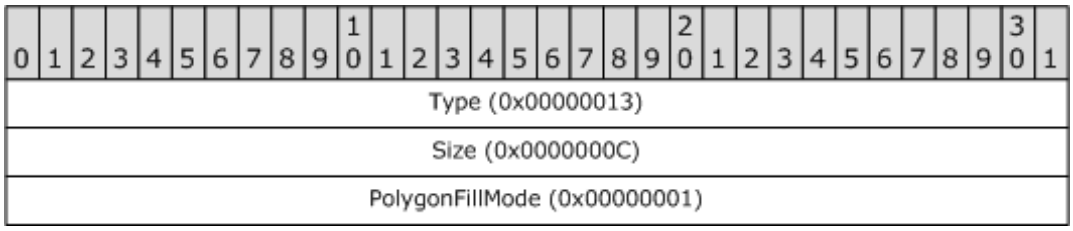


Figure 25: EMF EMR_SETPOLYFILLMODE Record Example

Type: 0x00000013 identifies the record type as EMR_SETPOLYFILLMODE.

Size: 0x0000000C specifies the size of this record in bytes.

PolygonFillMode: 0x00000001 specifies **ALTERNATE** polygon fill mode. This value is defined in the EMF PolygonFillMode enumeration.

3.2.18 EMR_SETROP2 Example 1

This section provides an example of the EMF EMR_SETROP2 record.

```
00000140:    14 00 00 00 0C 00 00 00 0D 00 00 00
```

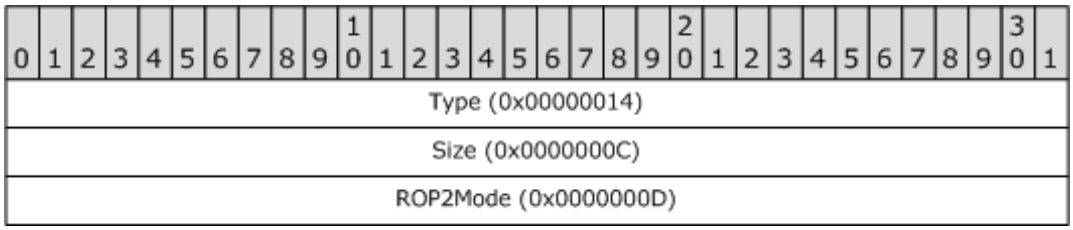


Figure 26: EMF EMR_SETROP2 Record Example

Type: 0x00000014 identifies the record type as EMR_SETROP2.

Size: 0x0000000C specifies the size of this record in bytes.

ROP2Mode: 0x0000000D specifies the **R2_COPYPEN** raster operation mode. This value is defined in the WMF Binary Raster Operation enumeration [MS-WMF].

3.2.19 EMR_SETSTRETCHBLTMODE Example 1

This section provides an example of the EMF EMR_SETSTRETCHBLTMODE record.

```
00000150: 15 00 00 00 0C 00 00 00 01 00 00 00
```

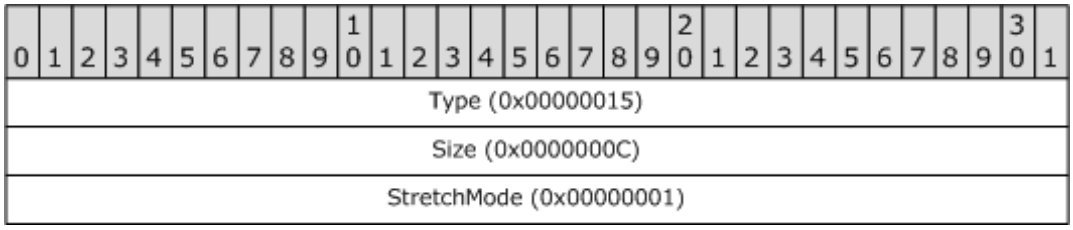


Figure 27: EMF EMR_SETSTRETCHBLTMODE Record Example

Type: 0x00000015 identifies the record type as EMR_SETSTRETCHBLTMODE.

Size: 0x0000000C specifies the size of this record in bytes.

StretchMode: 0x00000001 specifies a Boolean AND operation using the color values for the eliminated and existing pixels, from the EMF StretchMode enumeration.

3.2.20 EMR_SETTEXTALIGN Example 1

This section provides an example of the EMR_SETTEXTALIGN record.

```
00000150:                         16 00 00 00
00000160: 0C 00 00 00 00 00 00 00
```

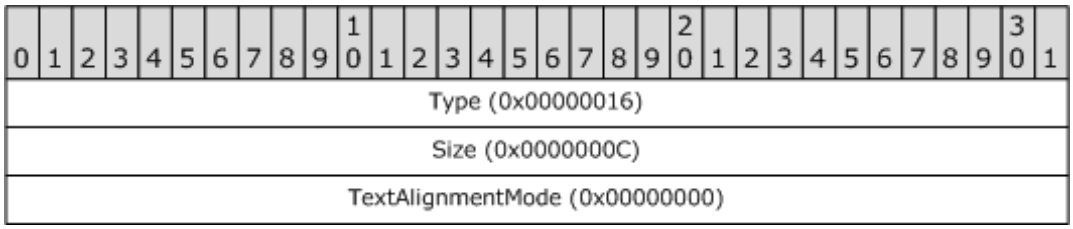


Figure 28: EMF EMR_SETTEXTALIGN Record Example

Type: 0x00000016 identifies this EMF record type as EMR_SETTEXTALIGN.

Size: 0x0000000C specifies the size of this record in bytes.

TextAlignmentMode: 0x00000000 specifies text alignment using a mask of WMF TextAlignmentMode flags or VerticalTextAlignmentMode flags [MS-WMF]. 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.

3.2.21 EMR_SETBRUSHORGEX Example 1

This section provides an example of the EMR_SETBRUSHORGEX record.

```
00000160:                         0D 00 00 00 10 00 00 00
00000170: 00 00 00 00 00 00 00 00
```

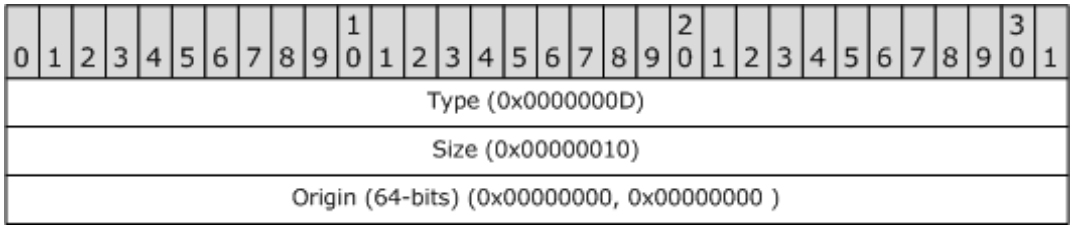


Figure 29: EMF EMR_SETBRUSHORGEX Record Example

Type: 0x0000000D identifies this EMF record type as EMR_SETBRUSHORGEX.

Size: 0x00000010 specifies the size of this record in bytes.

Origin: (0x00000000, 0x00000000) specifies the brush horizontal and vertical origin in device units.

3.2.22 EMR_SETMITERLIMIT Example 1

This section provides an example of the EMF EMR_SETMITERLIMIT record.

```
00000170:                         3A 00 00 00 0C 00 00 00
00000180: 0A 00 00 00
```

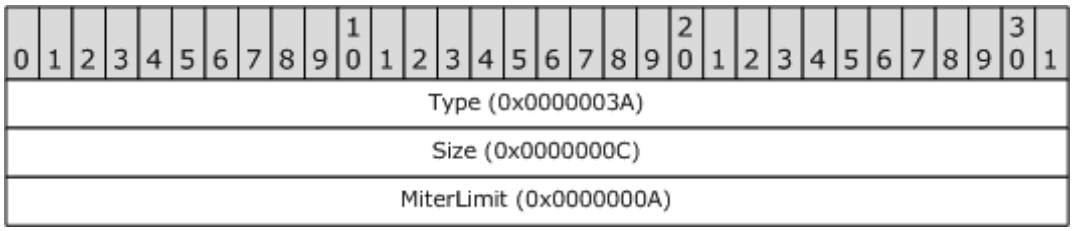


Figure 30: EMF EMR_SETMITERLIMIT Record Example

Type: 0x0000003A identifies the record type as EMR_SETMITERLIMIT.

Size: 0x0000000C specifies the size of this record in bytes.

MiterLimit: 0x0000000A specifies a miter length limit of 10 logical units.

3.2.23 EMR_MOVETOEX Example 1

This section provides an example of the EMR_MOVETOEX record.

```
00000180:             1B 00 00 00 10 00 00 00 00 00 00 00
00000190: 00 00 00 00
```

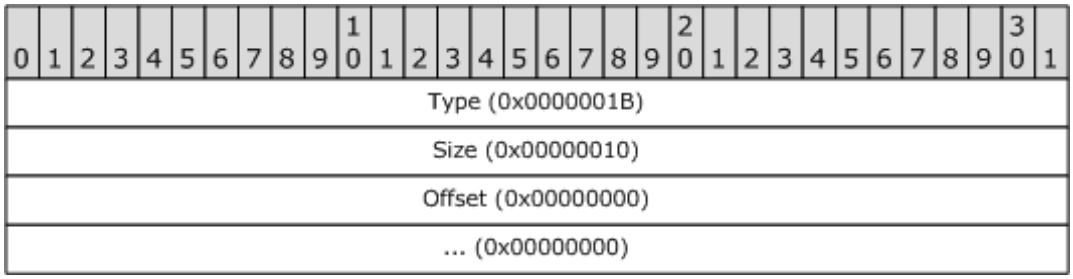


Figure 31: EMF EMR_MOVETOEX Record Example

Type: 0x0000001B identifies this EMF record type as EMR_MOVETOEX.

Size: 0x00000010 specifies the size of this record in bytes.

Offset: (0x00000000, 0x00000000) specifies coordinates of the new current position in logical units.

3.2.24 EMR_SETWORLDTRANSFORM Example 1

This section provides an example of the EMF EMR_SETWORLDTRANSFORM record.

```
00000190:             23 00 00 00 20 00 00 00 87 C3 81 3F
000001A0: 00 00 00 00 00 00 00 00 7A BD 80 3F 00 00 00 80
000001B0: 00 00 00 80
```

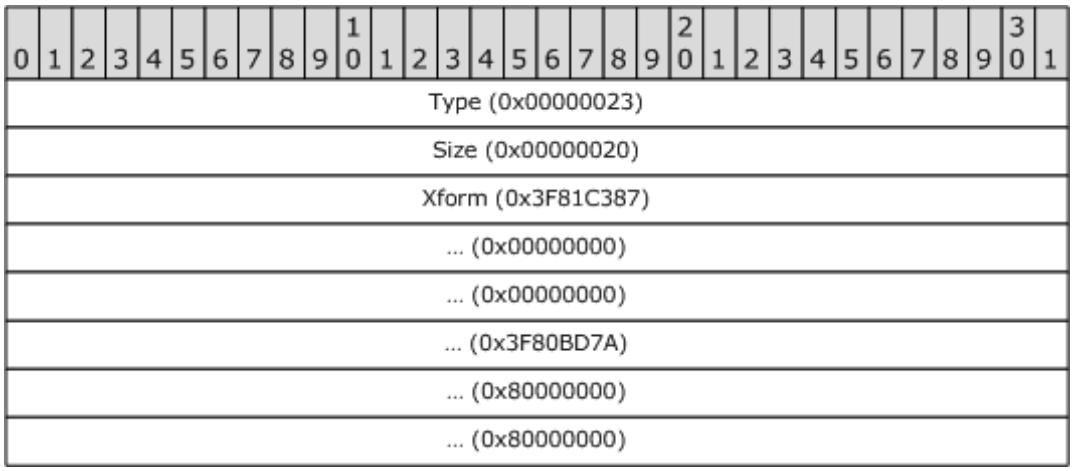


Figure 32: EMF EMR_SETWORLDTRANSFORM Record Example

Type: 0x00000023 identifies the record type as EMR_SETWORLDTRANSFORM.

Size: 0x00000020 specifies the size of this record in bytes.

Xform: (1.005782, 0.000000, 0.000000, 1.013780, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

3.2.25 EMR_MODIFYWORLDTRANSFORM Example 1

This section provides an example of the EMF EMR_MODIFYWORLDTRANSFORM record.

```
                        24 00 00 00 24 00 00 00 87 C3 81 3F
000001C0: 00 00 00 00 00 00 00 00 7A BD 80 3F 00 00 00 80
000001D0: 00 00 00 80 04 00 00 00
```

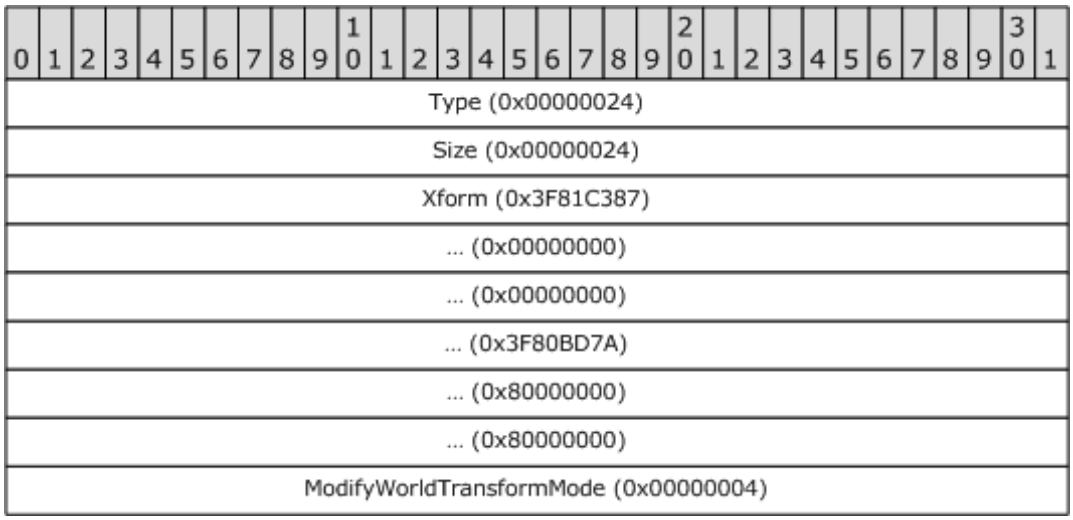


Figure 33: EMF EMR_MODIFYWORLDTRANSFORM Record Example

Type: 0x00000024 identifies the record type as EMR_MODIFYWORLDTRANSFORM.

Size: 0x00000024 specifies the size of this record in bytes.

Xform: (1.005782, 0.000000, 0.000000, 1.013780, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

ModifyWorldTransformMode: 0x00000004 specifies that this record performs the function of an EMF EMR_SETWORLDTRANSFORM record. This value is defined in the EMF ModifyWorldTransformMode enumeration.

3.2.26 EMR_SETLAYOUT Example 2

This section provides an example of the EMF EMR_SETLAYOUT record.

```
000001D0:                         73 00 00 00 0C 00 00 00
000001E0: 00 00 00 00
```

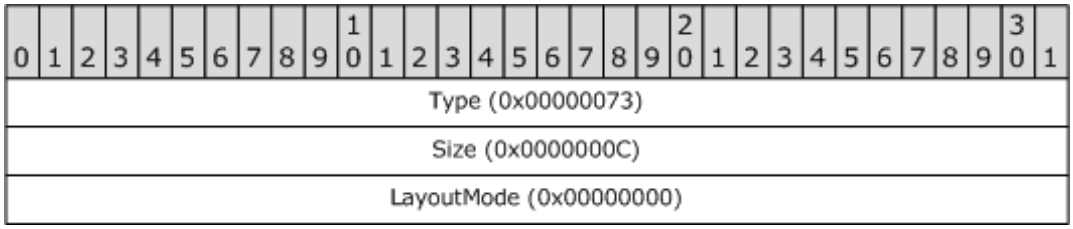


Figure 34: EMF EMR_SETLAYOUT Record Example

Type: 0x00000073 identifies this record type as EMR_SETLAYOUT.

Size: 0x0000000C specifies the size of this record in bytes.

LayoutMode: 0x00000000 specifies left-to-right horizontal layout.

3.2.27 EMR_SETBRUSHORGEX Example 2

This section provides an example of the EMR_SETBRUSHORGEX record.

```
000001E0:              0D 00 00 00 10 00 00 00 00 00 00 00
000001F0: 00 00 00 00
```

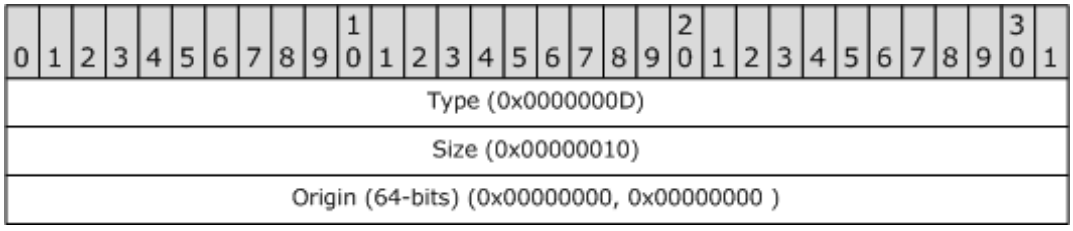


Figure 35: EMF EMR_SETBRUSHORGEX Record Example

Type: 0x0000000D identifies this EMF record type as EMR_SETBRUSHORGEX.

Size: 0x00000010 specifies the size of this record in bytes.

Origin: (0x00000000, 0x00000000) specifies the brush horizontal and vertical origin in device units.

3.2.28 EMR_EXTCREATEFONTINDIRECTW Example

This section provides an example of the EMF EMR_EXTCREATEFONTINDIRECTW record.

```
000001F0:             52 00 00 00 70 01 00 00 01 00 00 00
00000200: 10 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00
00000210: BC 02 00 00 00 00 00 00 07 02 02 22 53 00 79 00
00000220: 73 00 74 00 65 00 6D 00 00 00 00 00 00 00 00 00
00000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000260: B1 E6 31 7C BC FE 12 00 21 00 00 00 00 00 13 00
00000270: 00 00 2E 00 54 E1 12 00 BC FE 12 00 14 E1 12 00
00000280: 00 00 00 00 F4 F4 12 00 34 B8 F9 77 68 26 F4 77
00000290: FF FF FF FF AD 97 F7 77 7F 99 F7 77 70 E1 12 00
000002A0: 21 00 00 00 01 00 00 00 FC 8B CE 02 38 A6 17 00
000002B0: 7F 99 F7 77 B0 DA 4F 00 63 DE 5D 00 3E 08 5A 00
000002C0: 68 99 5A 00 F5 82 54 00 1A 91 54 00 8F 9B C7 77
000002D0: BD 93 54 00 6F 9D 54 00 94 E5 54 00 00 00 54 00
000002E0: AF 58 56 00 6B 50 56 00 B4 3A 55 00 05 39 55 00
000002F0: 09 35 55 00 64 C4 4F 00 00 82 41 00 FF 44 41 00
00000300: 12 4E 41 00 E1 4B 41 00 1D 1E 31 7C 4B 16 31 7C
00000310: DA EF 30 7C 49 F4 30 7C EA A3 37 7C 00 00 D5 77
00000320: A5 DC D5 77 46 46 D3 77 D7 96 D3 77 97 ED 31 7C
00000330: B1 E6 31 7C 00 00 2E 01 10 E2 12 00 68 8B CE 02
00000340: 08 ED F8 77 68 8B CE 02 00 00 2E 01 01 00 00 00
00000350: 00 00 2E 01 C4 04 F9 77 27 05 F9 77 64 76 00 08
00000360: 00 00 00 00
```

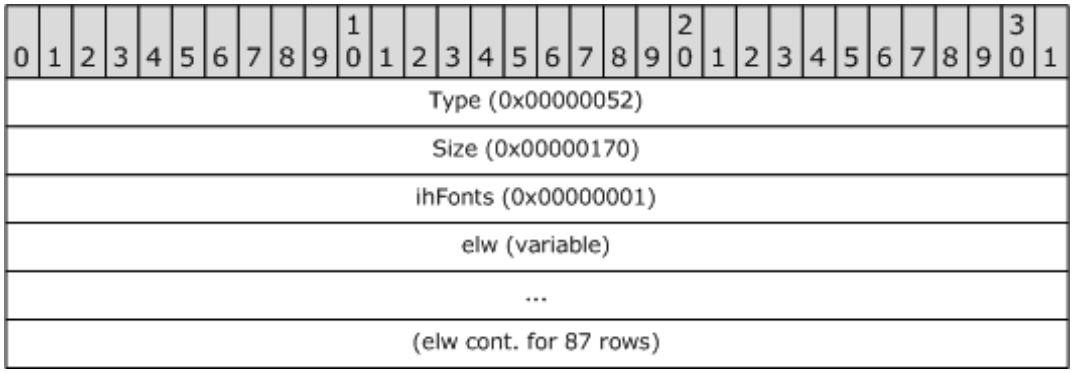


Figure 36: EMF EMR_EXTCREATEFONTINDIRECTW Record Example

Type: 0x00000052 identifies the record type as EMR_EXTCREATEFONTINDIRECTW.

Size: 0x00000170 specifies the size of this record in bytes.

ihFonts: 0x00000001 specifies the object index in the EMF Object Table to assign to the font.

elw: the logical font, which is an EMF LogFontExDv object.

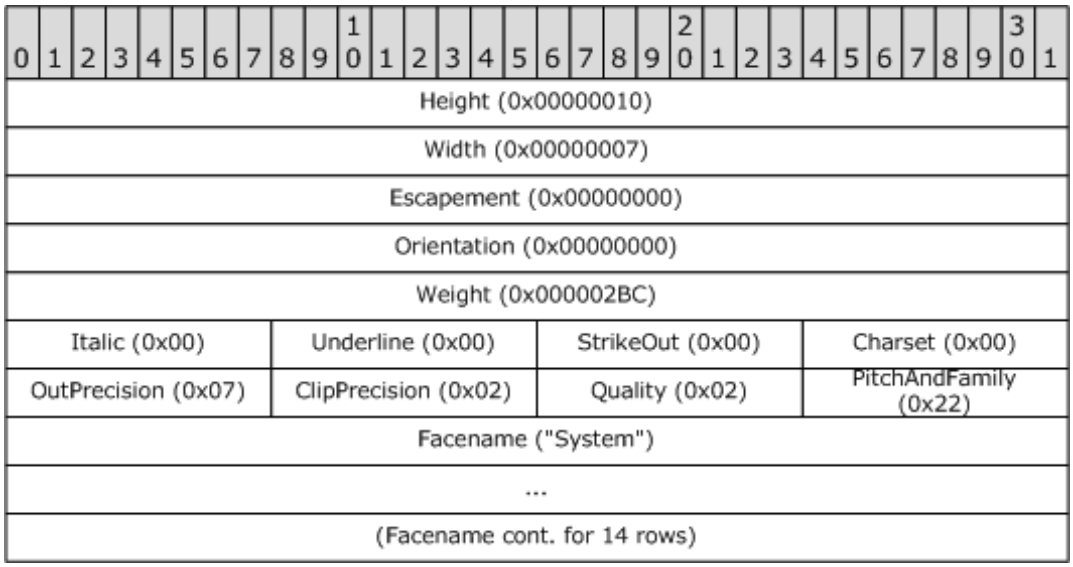


Figure 37: EMF LogFontExDv Object, Part 1

Height: 0x00000010 specifies the **cell height** for this font in logical units.

Width: 0x00000007 specifies the average character width for this font in logical units.

Escapement: 0x00000000 specifies an angle of 0 degrees between the baseline of a row of text and the x-axis of the device.

Orientation: 0x00000000 specifies an angle of 0 degrees between the baseline of each character and the x-axis of the device.

Weight: 0x0000002BC specifies that the **weight** of the font is 700, in the range 0 through 1000, from lightest to darkest, with 400 (0x00000190) considered normal.

Italic: 0x00 specifies that the font is not italic.

Underline: 0x00 specifies that the font is not underlined.

Strikeout: 0x00 specifies that the font characters do not have a strike-out graphic.

CharSet: 0x00 specifies the **ANSI_CHARSET**, as defined in the WMF CharacterSet enumeration [MS-WMF].

OutPrecision: 0x07 specifies the output precision, which is how closely the output matches the requested font properties, from the WMF OutPrecision enumeration. The value 0x07 specifies a **TrueType** font.

ClipPrecision: 0x02 specifies the clipping precision, which is how to clip characters that are partially outside the clipping region, from the WMF ClipPrecision Flags. The value 0x02 is used for vector and TrueType fonts.

Quality: 0x02 specifies proof output quality, from the WMF FontQuality enumeration.

PitchAndFamily: 0x22 specifies a variable-pitch font with no serifs, from the WMF FamilyFont and PitchFont enumerations.

Facename: "System" specifies the typeface name of the font in Unicode characters.

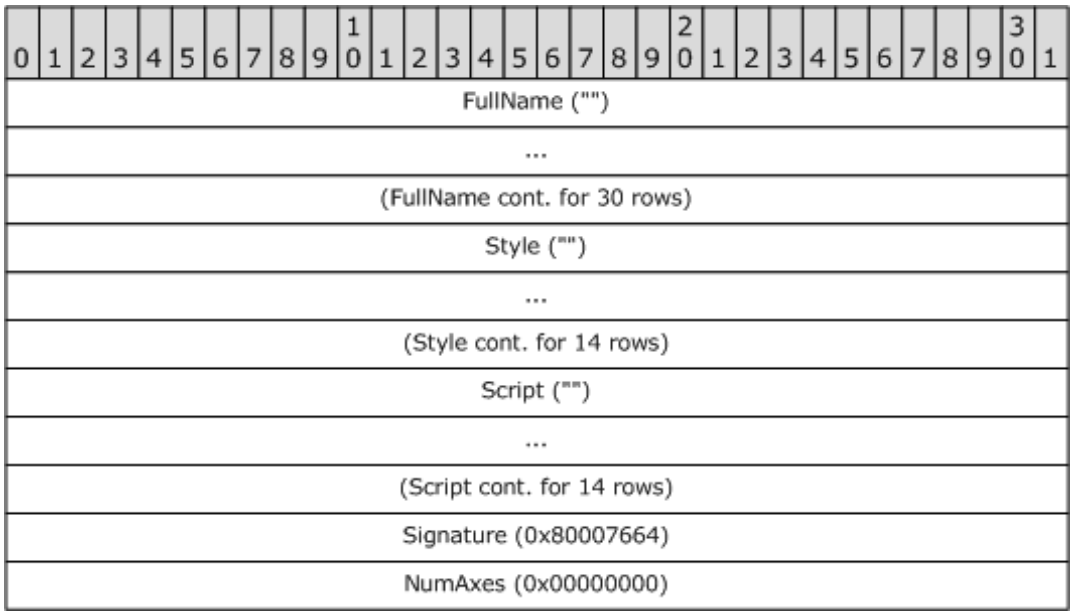


Figure 38: EMF LogFontExDv Object, Part 2

FullName: An empty string specifies the full name of the font.

Style: An empty string describes the style of the font.

Script: An empty string describes the character set of the font.

Signature: 0x80007664 specifies the signature of an EMF DesignVector object.

NumAxes: 0x00000000 specifies the number of **font axes** described in the DesignVector object.

3.2.29 EMR_SELECTOBJECT Example 4

This section provides an example of the EMR_SELECTOBJECT record.

```
00000360:     25 00 00 00 0C 00 00 00 01 00 00 00
```

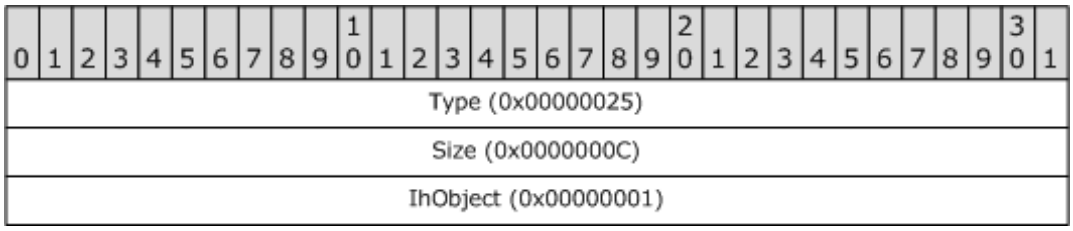


Figure 39: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x00000001 specifies the index of an object in the object table.

3.2.30 EMR_SELECTOBJECT Example 5

This section provides an example of the EMR_SELECTOBJECT record.

```
00000370: 25 00 00 00 0C 00 00 00 0E 00 00 80
```

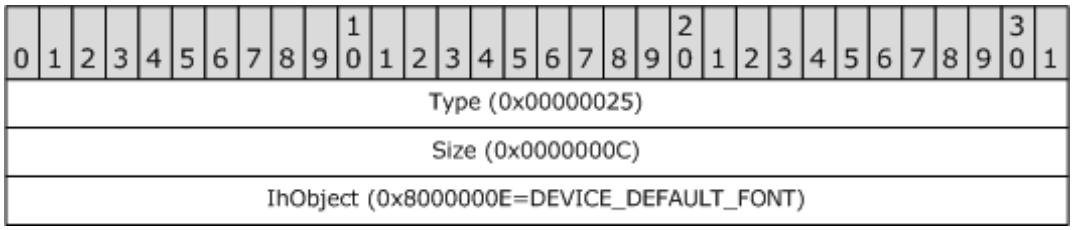


Figure 40: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x8000000E specifies the index of an object in the object table.

3.2.31 EMR_DELETEOBJECT Example

This section provides an example of the EMR_DELETEOBJECT record.

```
00000370:                         28 00 00 00
00000380: 0C 00 00 00 01 00 00 00
```

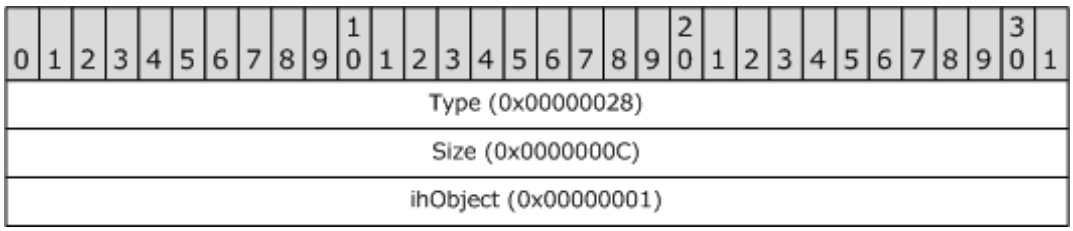


Figure 41: EMF EMR_DELETEOBJECT Record Example

Type: 0x00000028 identifies this EMF record type as EMR_DELETEOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x00000001 specifies the object table to be deleted.

3.2.32 EMR_COMMENT_EMFPLUS Example 2

This section provides an example of the EMF EMR_COMMENT_EMFPLUS record.

```
00000380:                         46 00 00 00 88 02 00 00
00000390: 7C 02 00 00 45 4D 46 2B
```

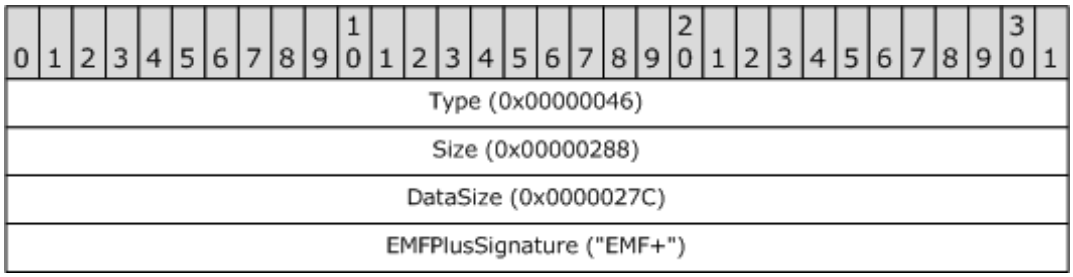


Figure 42: EMF EMR_COMMENT_EMFPLUS Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x00000288 specifies the size of this record in bytes, including embedded EMF+ records.

DataSize: 0x0000027C specifies the size of the EMF+ data in bytes.

EMFPlusSignature: "EMF+" identifies the comment record type as EMR_COMMENT_EMFPLUS.

The embedded EMF+ records are presented in the sections that follow.

3.2.32.1 EmfPlusSetAntiAliasMode Example 1

This section provides an example of the EmfPlusAntiAliasMode record.

```
00000390:                   1E 40 0B 00 0C 00 00 00
000003A0: 00 00 00 00
```

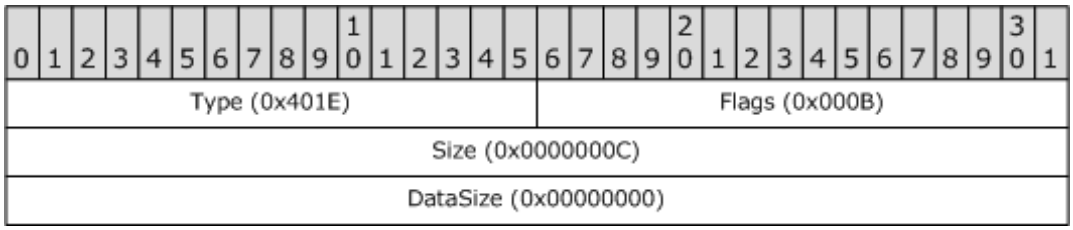


Figure 43: EmfPlusSetAntiAliasMode Record Example

Type: 0x401E identifies this record type as EmfPlusAntiAliasMode.

Flags: 0x000B specifies anti-aliasing with an 8x8 box filter.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.2 EmfPlusSetCompositingQuality Example 1

This section provides an example of the EmfPlusSetCompositingQuality record.

```
000003A0:            24 40 02 00 0C 00 00 00 00 00 00 00
```

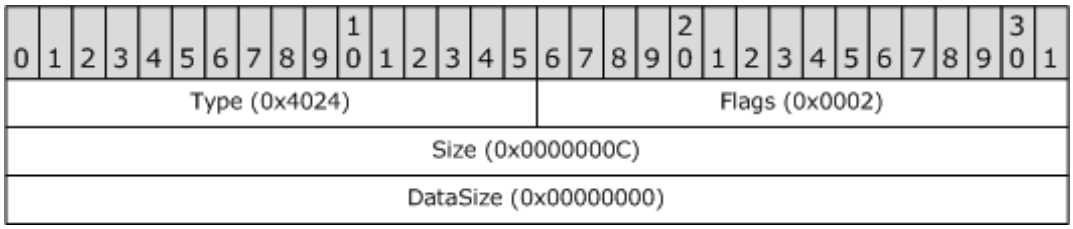


Figure 44: EmfPlusSetCompositingQuality Record Example

Type: 0x4024 identifies this record type as EmfPlusSetCompositingQuality.

Flags: 0x0002 specifies CompositingQualityHighSpeed from the CompositingQuality enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.3 EmfPlusSetInterpolationMode Example 1

This section provides an example of the EmfPlusSetInterpolationMode record.

```
000003B0: 21 40 07 00 0C 00 00 00 00 00 00 00
```

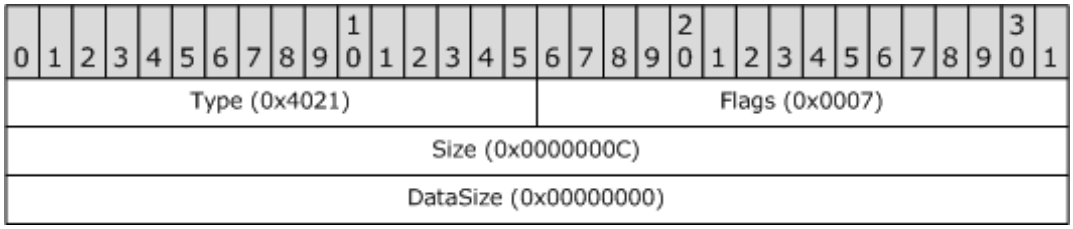


Figure 45: EmfPlusSetInterpolationMode Record Example

Type: 0x4021 identifies this record type as EmfPlusSetInterpolationMode.

Flags: 0x0007 specifies InterpolationModeHighQualityBicubic from the InterpolationMode enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.4 EmfPlusSetPixelOffsetMode Example 1

This section provides an example of the EmfPlusSetPixelOffsetMode record.

```
000003B0:                         22 40 03 00
000003C0: 0C 00 00 00 00 00 00 00
```

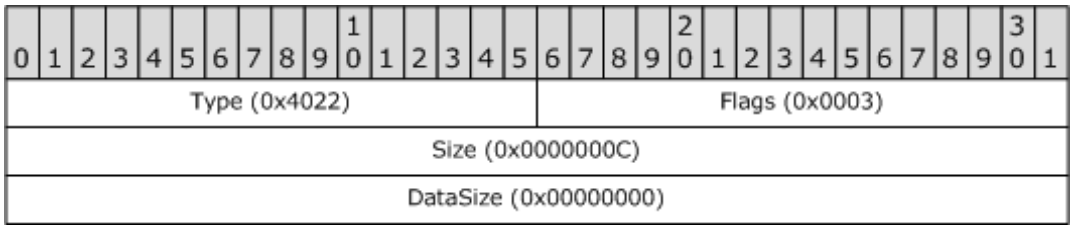


Figure 46: EmfPlusSetPixelOffsetMode Record Example

Type: 0x4022 identifies this record type as EmfPlusSetPixelOffsetMode.

Flags: 0x0003 specifies PixelOffsetModeNone from the PixelOffsetMode enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.5 EmfPlusSetTextRenderingHint Example 1

This section provides an example of the EmfPlusSetTextRenderingHint record.

```
000003C0:              1F 40 05 00 0C 00 00 00
000003D0: 00 00 00 00
```

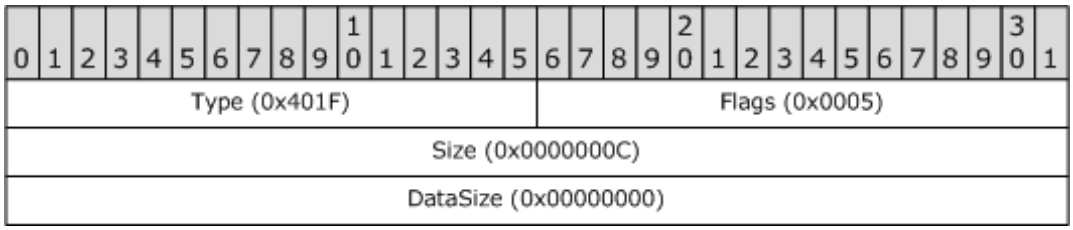


Figure 47: EmfPlusSetTextRenderingHint Record Example

Type: 0x401F identifies this record type as EmfPlusSetTextRenderingHint.

Flags: 0x0005 specifies TextRenderingHintClearTypeGridFit from the TextRenderingHint enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case, there is none.

3.2.32.6 EmfPlusMultiplyWorldTransform Example 1

This section provides an example of the EmfPlusMultiplyWorldTransform record.

```
000003D0:              2C 40 00 00 24 00 00 00 18 00 00 00
000003E0: 00 00 80 3F 00 00 00 00 00 00 00 00 00 00 80 3F
000003F0: 00 00 00 80 00 00 00 80
```

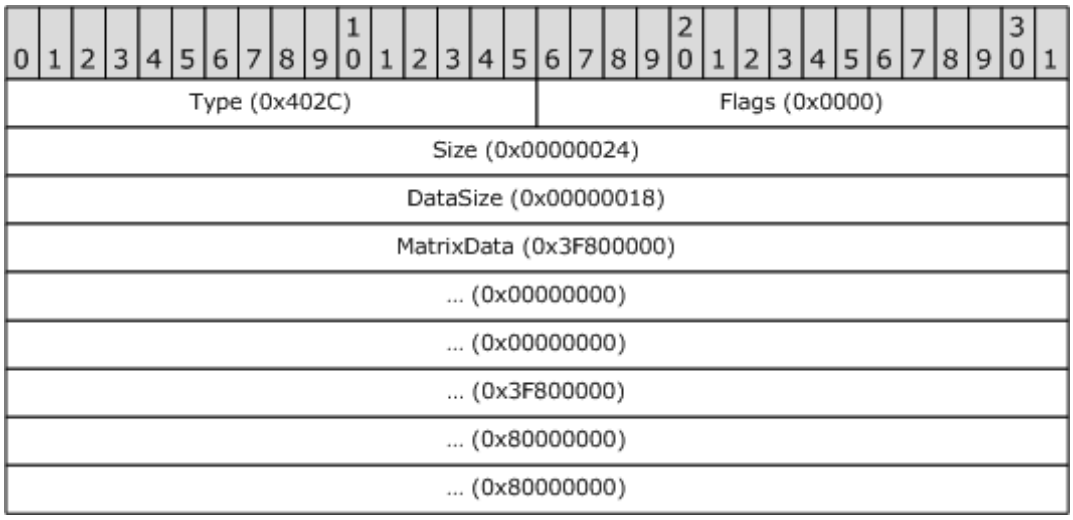


Figure 48: EmfPlusMultiplyWorldTransform Record Example

Type: 0x402C identifies the record type as EmfPlusMultiplyWorldTransform.

Flags: 0x0000 specifies pre-multiplication of the transform matrix.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the record-specific data that follows.

MatrixData: An EmfPlusTransformMatrix object that contains the multiplication matrix.

3.2.32.7 EmfPlusSave Example

This section provides an example of the EmfPlusSave record.

```
000003F0:                         25 40 00 00 10 00 00 00
00000400: 04 00 00 00 00 00 00 00
```

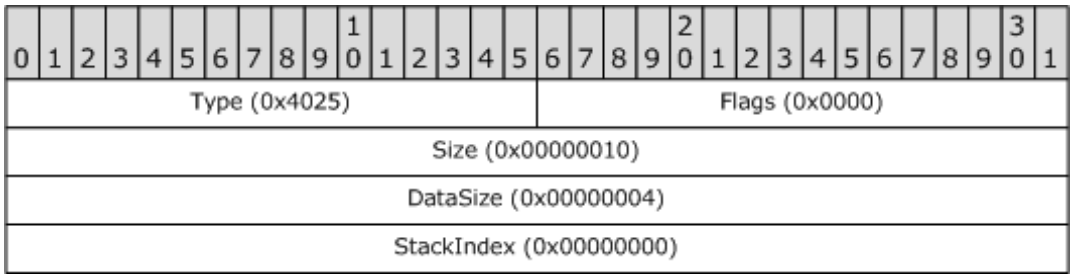


Figure 49: EmfPlusSave Record Example

Type: 0x4025 identifies this record type as EmfPlusSave.

Flags: 0x0000 This field is undefined for this record type.

Size: 0x00000010 specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

DataSize: 0x00000004 specifies the 32-bit-aligned number of bytes of data in the **PointData** member that follows. This number does not include the size of the invariant part of this record.

StackIndex: 0x00000000 specifies the identifier used by the corresponding EmfPlusRestore operation to retrieve the graphics state from the correct save level on the graphics state stack.

3.2.32.8 EmfPlusMultiplyWorldTransform Example 2

This section provides an example of the EmfPlusMultiplyWorldTransform record.

```
00000400:                         2C 40 00 00 24 00 00 00
00000410: 18 00 00 00 00 00 80 3F 00 00 00 00 00 00 00 00
00000420: 00 00 80 3F 00 00 00 00 00 00 00 00
```

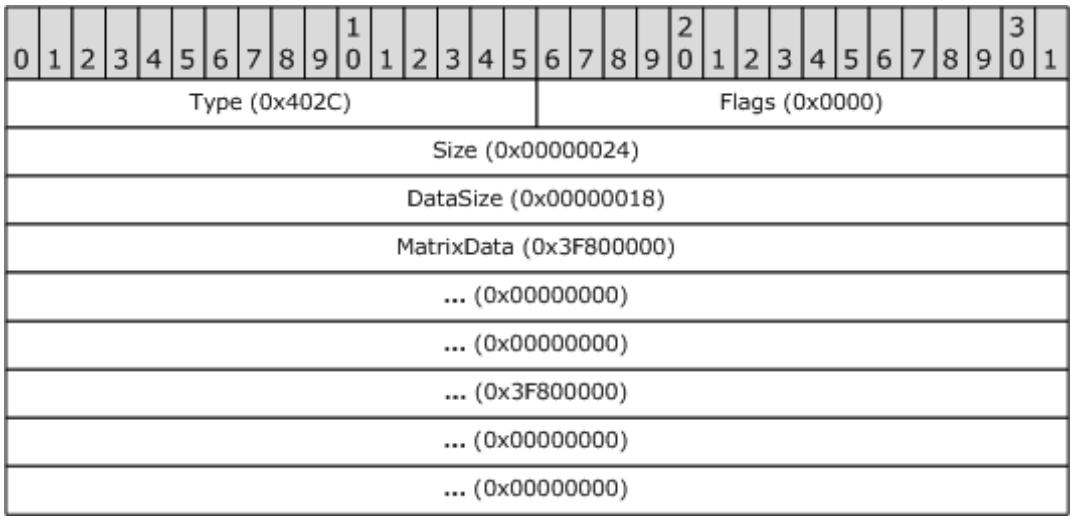


Figure 50: EmfPlusMultiplyWorldTransform Record Example

Type: 0x402C identifies the record type as EmfPlusMultiplyWorldTransform.

Flags: 0x0000 specifies pre-multiplication of the transform matrix.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the record-specific data that follows.

MatrixData: An EmfPlusTransformMatrix object that contains the multiplication matrix.

3.2.32.9 EmfPlusSetWorldTransform Example 1

This section provides an example of the EmfPlusSetWorldTransform record.

```
00000420:                                     2A 40 00 00
00000430: 24 00 00 00 18 00 00 00 00 00 80 3F 00 00 00 00
00000440: 00 00 00 00 00 00 80 3F 00 00 00 80 00 00 00 80
```

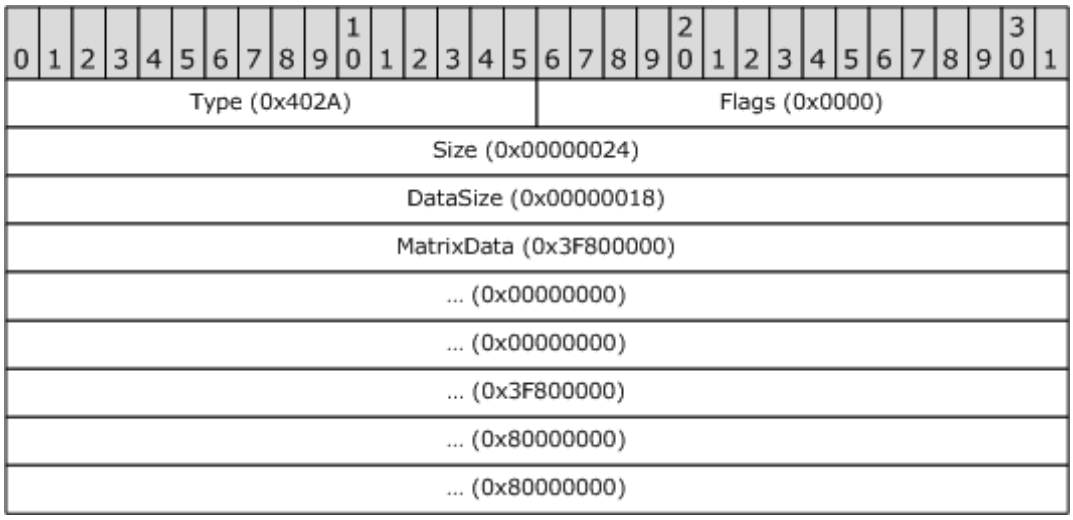


Figure 51: EmfPlusSetWorldTransform Record Example

Type: 0x402A identifies this record type as EmfPlusSetWorldTransform.

Flags: 0x0000 This field is undefined for this record type.

Size: 0x00000024 specifies the size in bytes of this record.

DataSize: 0x00000018 specifies the size in bytes of record-specific data in this record.

MatrixData: An EmfPlusTransformMatrix object, which specifies the world transform.

3.2.32.10 EmfPlusBeginContainerNoParams Example

This section provides an example of the EmfPlusBeginContainerNoParams record.

```
00000450: 28 40 00 00 10 00 00 00 04 00 00 00 01 00 00 00
```

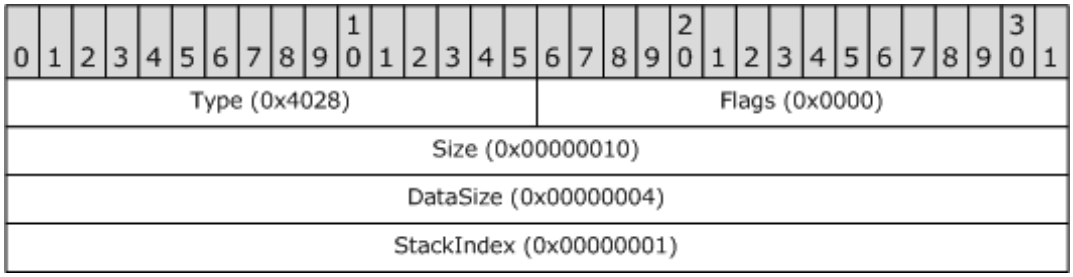


Figure 52: EmfPlusBeginContainerNoParams Record Example

Type: 0x4028 identifies the record type as EmfPlusBeginContainerNoParams.

Flags: 0x0000 is undefined for this record type.

Size: 0x00000010 specifies the size in bytes of the entire record.

DataSize: 0x00000004 specifies the size in bytes of the record-specific data that follows.

StackIndex: 0x00000001 specifies the identifier used to reference the container in future records. It will match a **StackIndex** value in a subsequent EmfPlusEndContainer record that will end the container.

3.2.32.11 EmfPlusSetAntiAliasMode Example 2

This section provides an example of the EmfPlusAntiAliasMode record.

```
00000460: 1E 40 0B 00 0C 00 00 00 00 00 00 00
```

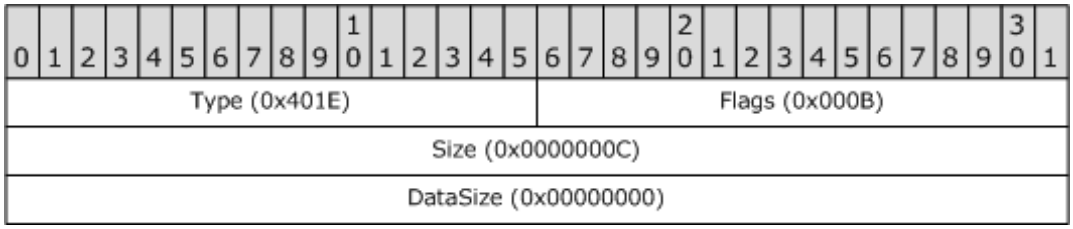


Figure 53: EmfPlusSetAntiAliasMode Record Example

Type: 0x401E identifies this record type as EmfPlusAntiAliasMode.

Flags: 0x000B specifies anti-aliasing with an 8x8 box filter.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.12 EmfPlusSetCompositingQuality Example 2

This section provides an example of the EmfPlusSetCompositingQuality record.

```
00000460:                         24 40 02 00
00000470: 0C 00 00 00 00 00 00 00
```

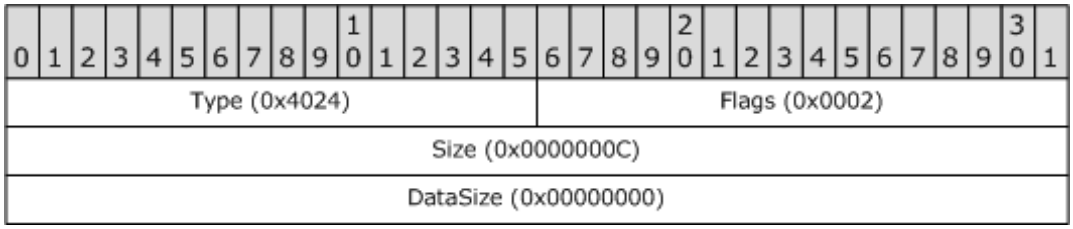


Figure 54: EmfPlusSetCompositingQuality Record Example

Type: 0x4024 identifies this record type as EmfPlusSetCompositingQuality.

Flags: 0x0002 specifies CompositingQualityHighSpeed from the CompositingQuality enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.13 EmfPlusSetInterpolationMode Example 2

This section provides an example of the EmfPlusSetInterpolationMode record.

```
00000470:                       21 40 07 00 0C 00 00 00
00000480: 00 00 00 00
```

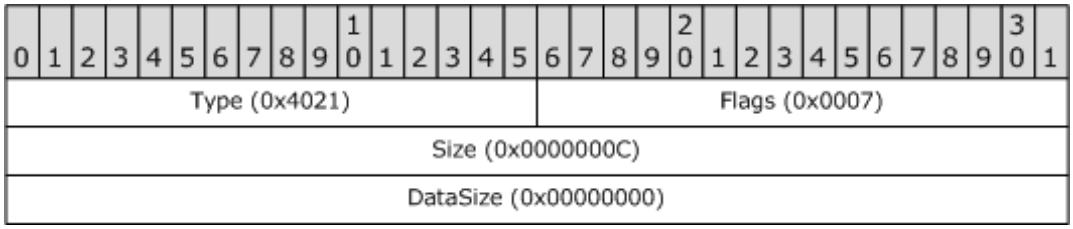


Figure 55: EmfPlusSetInterpolationMode Record Example

Type: 0x4021 identifies this record type as EmfPlusSetInterpolationMode.

Flags: 0x0007 specifies InterpolationModeHighQualityBicubic from the InterpolationMode enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.14 EmfPlusSetPixelOffsetMode Example 2

This section provides an example of the EmfPlusSetPixelOffsetMode record.

```
00000480:             22 40 03 00 0C 00 00 00 00 00 00 00
```

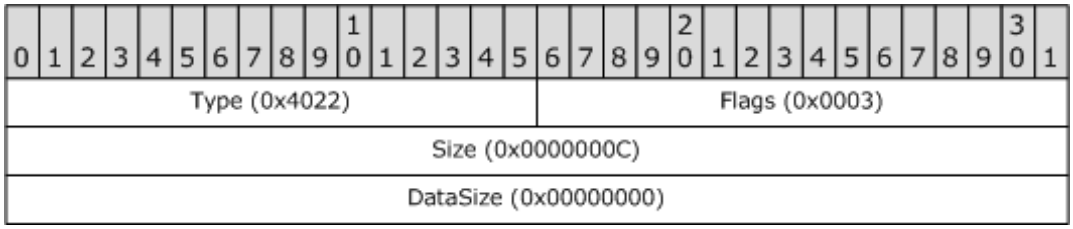


Figure 56: EmfPlusSetPixelOffsetMode Record Example

Type: 0x4022 identifies this record type as EmfPlusSetPixelOffsetMode.

Flags: 0x0003 specifies PixelOffsetModeNone from the PixelOffsetMode enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case there is none.

3.2.32.15 EmfPlusSetTextRenderingHint Example 2

This section provides an example of the EmfPlusSetTextRenderingHint record.

```
00000490: 1F 40 05 00 0C 00 00 00 00 00 00 00
```

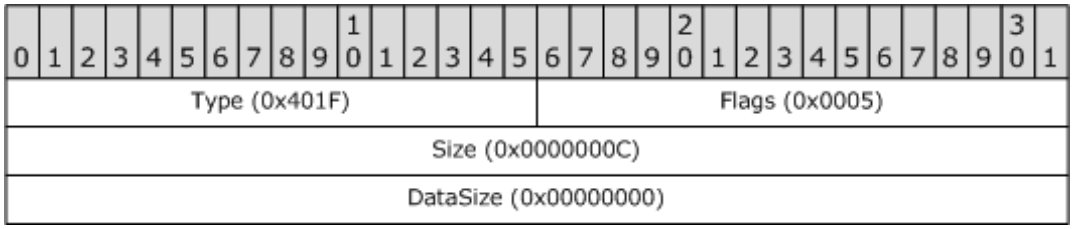


Figure 57: EmfPlusSetTextRenderingHint Record Example

Type: 0x401F identifies this record type as EmfPlusSetTextRenderingHint.

Flags: 0x0005 specifies TextRenderingHintClearTypeGridFit from the TextRenderingHint enumeration.

Size: 0x0000000C specifies the size in bytes of this record.

DataSize: 0x00000000 specifies the size in bytes of record-specific data in this record, and in this case, there is none.

3.2.32.16 EmfPlusSetPageTransform Example

This section provides an example of the EmfPlusSetPageTransform record.

```
00000490:                                     30 40 02 00
000004A0: 10 00 00 00 04 00 00 00 00 00 80 3F
```

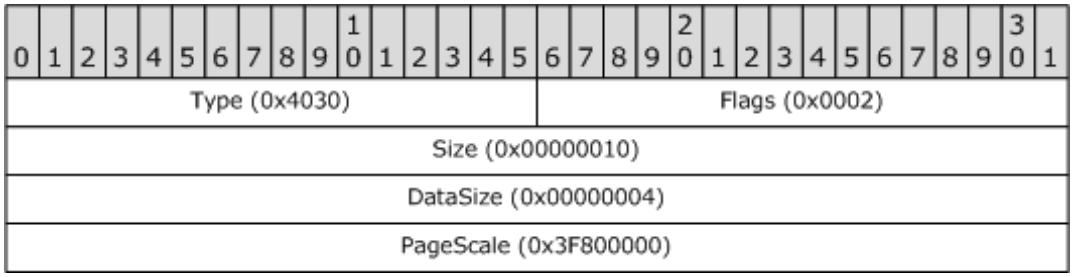


Figure 58: EmfPlusSetPageTransform Record Example

Type: 0x4030 identifies this record type as EmfPlusSetPageTransform.

Flags: 0x0002 specifies UnitPixel from the UnitType enumeration.

Size: 0x00000010 specifies the size in bytes of this record.

DataSize: 0x00000004 specifies the size in bytes of the record-specific data in this record.

PageScale: 0x3F800000 specifies the floating-point scale factor for converting page space coordinates to device space coordinates.

3.2.32.17 EmfPlusSetWorldTransform Example 2

This section provides an example of the EmfPlusSetWorldTransform record.

```
000004A0:                                       2A 40 00 00
000004B0: 24 00 00 00 18 00 00 00 00 00 80 3F 00 00 00 00
000004C0: 00 00 00 00 00 00 80 3F 00 00 00 80 00 00 00 80
```

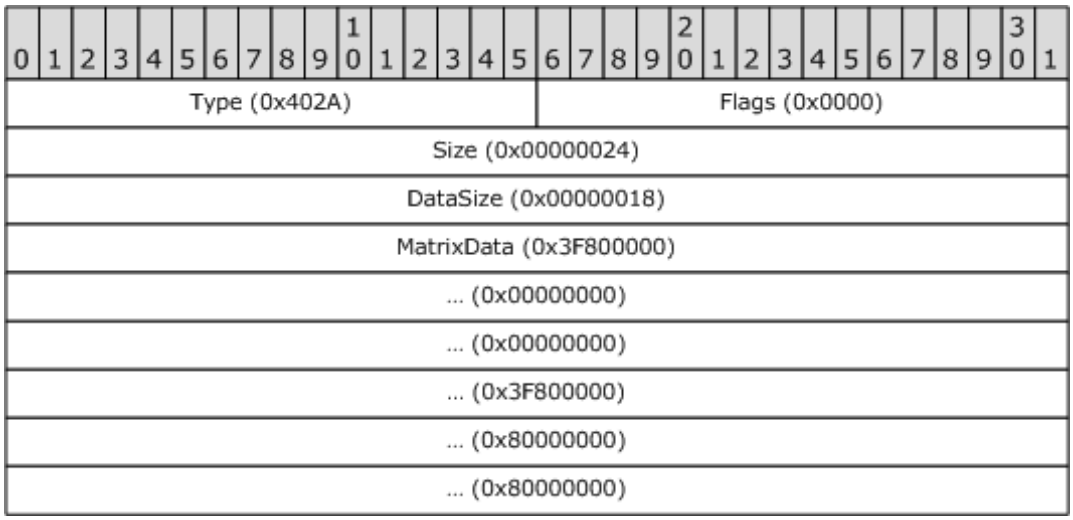


Figure 59: EmfPlusSetWorldTransform Record Example

Type: 0x402A identifies the record type as EmfPlusSetWorldTransform.

Flags: 0x0000 is undefined for this record type.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the **MatrixData** field that follows.

MatrixData: An EmfPlusTransformMatrix object, which contains the world-space transform.

3.2.32.18 EmfPlusSetWorldTransform Example 3

This section provides an example of the EmfPlusSetWorldTransform record.

```
000004D0: 2A 40 00 00 24 00 00 00 18 00 00 00 00 00 80 3F
000004E0: 00 00 00 00 00 00 00 00 00 00 80 3F 00 00 00 80
000004F0: 00 00 00 80
```

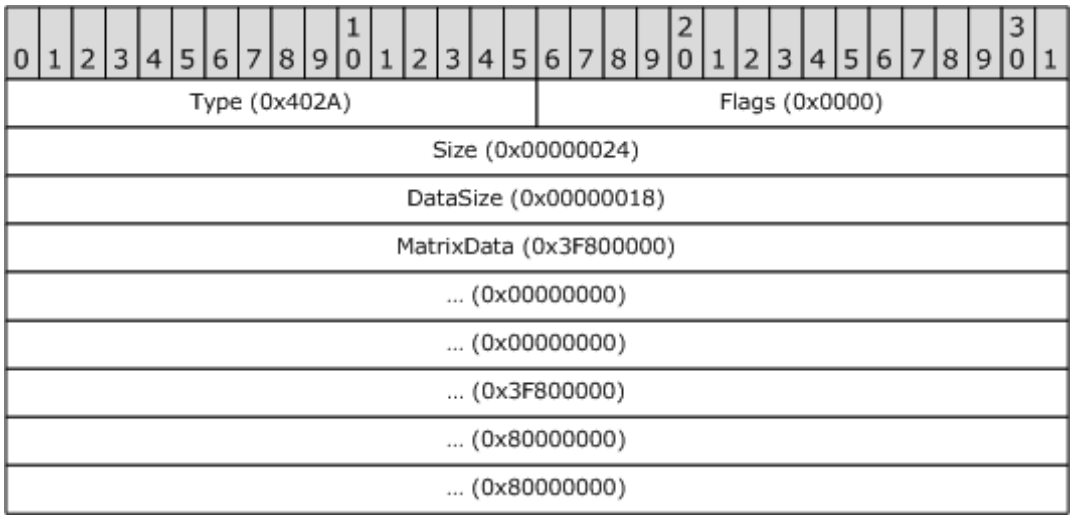


Figure 60: EmfPlusSetWorldTransform Record Example

Type: 0x402A identifies the record type as EmfPlusSetWorldTransform.

Flags: 0x0000 is undefined for this record type.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the **MatrixData** field that follows.

MatrixData: An EmfPlusTransformMatrix object that contains the world-space transform.

3.2.32.19 EmfPlusSetWorldTransform Example 4

This section provides an example of the EmfPlusSetWorldTransform record.

```
000004F0:             2A 40 00 00 24 00 00 00 18 00 00 00
00000500: 0D 74 DA 3A 00 00 00 00 00 00 00 00 0D 74 DA 3A
00000510: 00 00 00 80 00 00 00 80
```

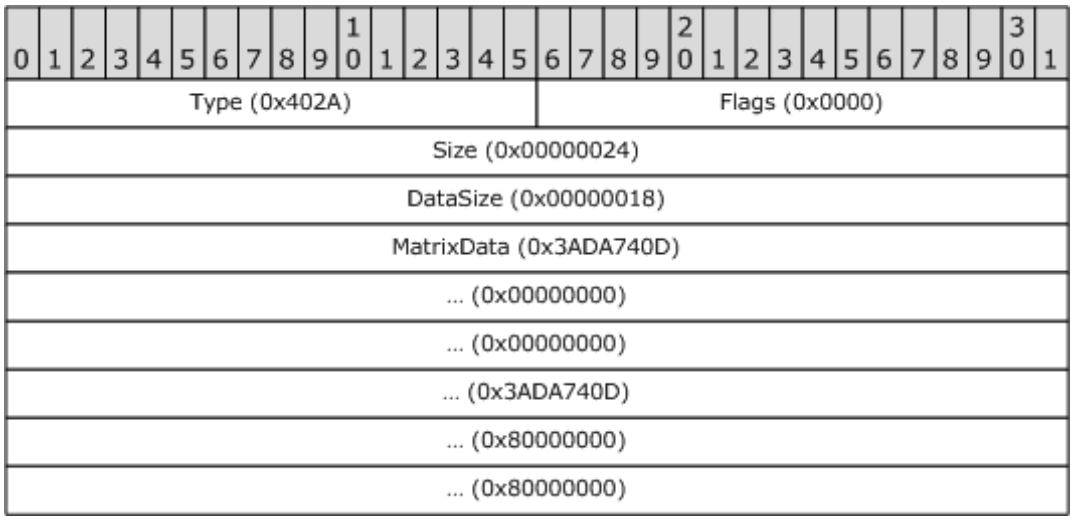


Figure 61: EmfPlusSetWorldTransform Record Example

Type: 0x402A identifies the record type as EmfPlusSetWorldTransform.

Flags: 0x0000 is undefined for this record type.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the **MatrixData** field that follows.

MatrixData: An EmfPlusTransformMatrix object, which contains the world-space transform.

3.2.32.20 EmfPlusSetWorldTransform Example 5

This section provides an example of the EmfPlusSetWorldTransform record.

```
00000510:                         2A 40 00 00 24 00 00 00
00000520: 18 00 00 00 92 5F 2C 3E 00 00 00 00 00 00 00 00
00000530: EB 51 38 3E 00 00 00 80 00 00 00 80
```

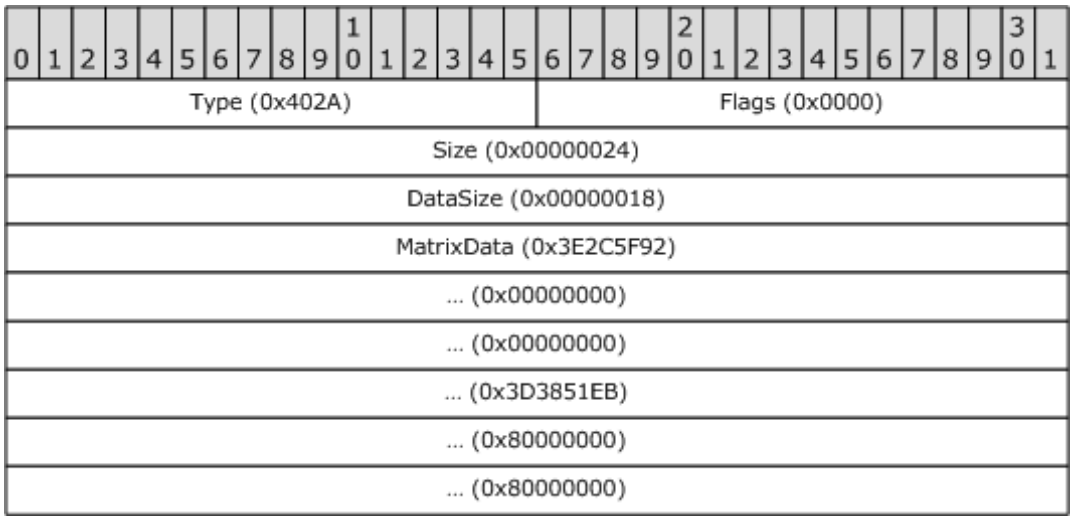


Figure 62: EmfPlusSetWorldTransform Record Example

Type: 0x402A identifies the record type as EmfPlusSetWorldTransform.

Flags: 0x0000 is undefined for this record type.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the **MatrixData** field that follows.

MatrixData: An EmfPlusTransformMatrix object, which contains the world-space transform.

3.2.32.21 EmfPlusObject Example

This section provides an example of the EmfPlusObject record.

```
00000530:                                     08 40 00 03
00000540: C4 00 00 00 B8 00 00 00 02 10 C0 DB 13 00 00 00
00000550: 00 00 00 00 FC 7F F5 43 CF FF FF BF 9D 8E 08 44
00000560: 1E 01 00 C0 FE BF 13 44 EB 15 2B 42 FF BF 13 44
00000570: FC FF C3 42 FF BF 13 44 FE FF C3 42 FF BF 13 44
00000580: 01 00 C4 42 FF BF 13 44 03 00 C4 42 FF BF 13 44
```

```
00000590: FF FF F5 43 FF BF 13 44 9F CE 08 44 9F 8E 08 44
000005A0: FF FF 13 44 00 80 F5 43 FF FF 13 44 05 00 C2 42
000005B0: FF FF 13 44 16 16 27 42 00 00 14 44 72 FF 3F C0
000005C0: 9F CE 08 44 E8 FF 3F C0 01 00 F6 43 10 00 40 C0
000005D0: 04 00 C4 42 64 00 40 C0 17 16 2B 42 FA 15 27 42
000005E0: E8 FE FF BF F6 FF C1 42 26 00 00 C0 00 03 03 03
000005F0: 03 03 03 01 03 03 03 01 03 03 03 01 03 03 83 BF
```

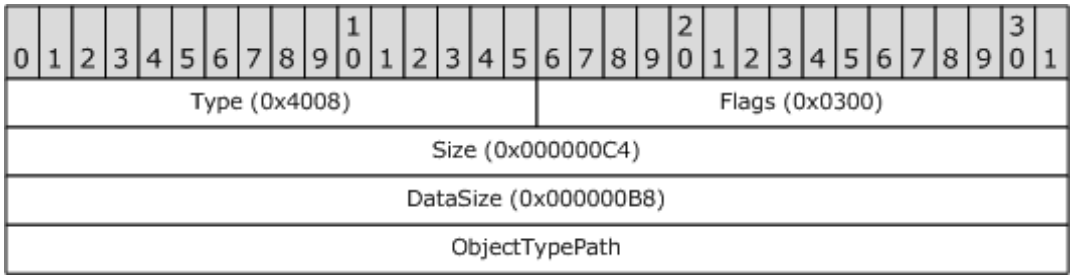


Figure 63: EmfPlusObject Record Example

Type: 0x4008 identifies the record type as EmfPlusObject.

Flags: 0x0300 specifies an EmfPlusPath object from the ObjectType enumeration, and index 0x00 for the graphics object in the EMF+ Object Table.

Size: 0x000000C4 specifies the size in bytes of the entire record.

DataSize: 0x000000B8 specifies the size in bytes of the record-specific data that follows.

ObjectTypePath: An EmfPlusPath object.

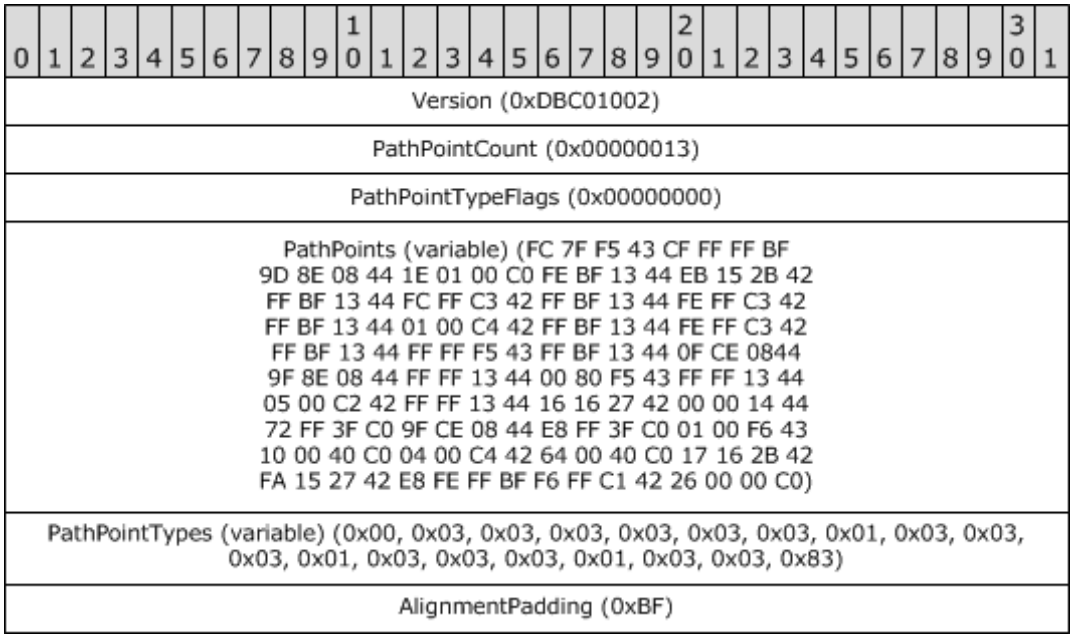


Figure 64: EmfPlusPath Object Example

Version: 0xDBC01002 specifies the graphics version that was used to create the EmfPlusPath object in this record. The value 0xDBC01002 corresponds to GDI+ version 1.1.

PathPointCount: 0x00000013 specifies the number of elements in the **PathPoints** and **PathPointTypes** arrays.

PathPointTypeFlags: 0x00000000 specifies that the path point type values are not compressed.

PathPoints: An array of **PathPointCount** EmfPlusPointF objects that specify the endpoints and control points of the lines and Bezier curves that define the path.

PathPointTypes: An array of **PathPointCount** bytes that specify the point types and flags for the data points in the path. Point types determine how the points are used to draw the path, and are values in the PathPointType enumeration.

AlignmentPadding: An extra byte that makes the total size of this record a multiple of 4 bytes. The value of this field is indeterminate and is ignored.

3.2.32.22 EmfPlusFillPath Example

This section provides an example of the EmfPlusFillPath record.

```
00000600: 14 40 00 80 10 00 00 00 04 00 00 00 FF FF FF 00
```

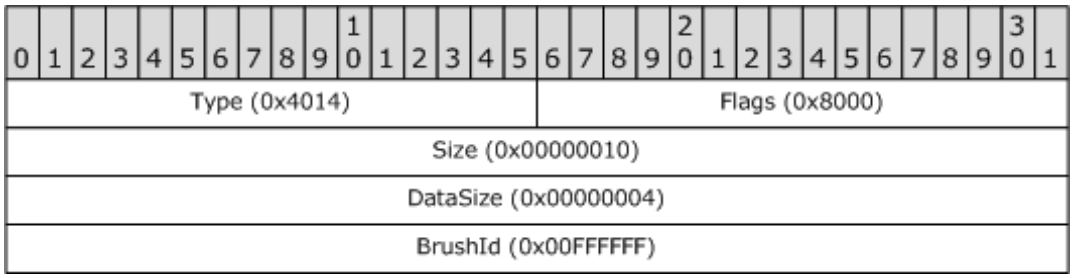


Figure 65: EmfPlusFillPath Record Example

Type: 0x4014 identifies the record type as EmfPlusFillPath.

Flags: 0x8000 specifies that a solid color is specified in the **BrushId** field.

Size: 0x00000010 specifies the size in bytes of the entire record.

DataSize: 0x00000004 specifies the size in bytes of the **BrushID** field that follows.

BrushId: 0x00FFFFFF specifies the fill color, which is white.

3.2.33 EMR_COMMENT_EMFPLUS Example 3

This section provides an example of the EMF EMR_COMMENT_EMFPLUS record.

```
00000610: 46 00 00 00 50 00 00 00 44 00 00 00 45 4D 46 2B
```

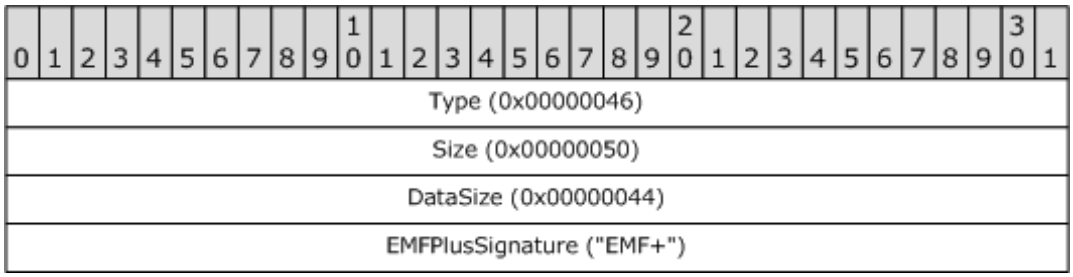


Figure 66: EMF EMR_COMMENT_EMFPLUS Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x00000050 specifies the size of this record in bytes, including embedded EMF+ records.

DataSize: 0x00000044 specifies the size of the EMF+ data in bytes.

EMFPlusSignature: "EMF+" identifies the comment record type as EMR_COMMENT_EMFPLUS.

The embedded EMF+ records are presented in the sections that follow.

3.2.33.1 EmfPlusObject Example

This section provides an example of the EmfPlusObject record.

```
00000620: 08 40 01 02 30 00 00 00 24 00 00 00 02 10 C0 DB
00000630: 00 00 00 00 80 00 00 00 02 00 00 00 00 00 80 3F
00000640: 00 00 00 00 02 10 C0 DB 00 00 00 00 00 00 00 FF
```

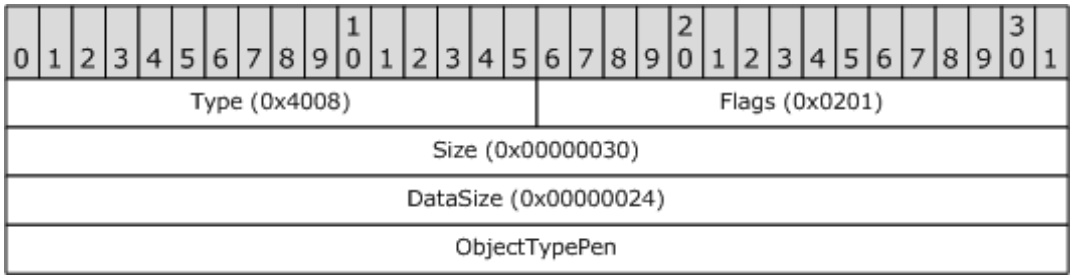


Figure 67: EmfPlusObject Record Example

Type: 0x4008 identifies the record type as EmfPlusObject.

Flags: 0x0201 specifies an EmfPlusPen object from the ObjectType enumeration, and index 0x01 for the graphics object in the EMF+ Object Table.

Size: 0x00000030 specifies the size in bytes of the entire record.

DataSize: 0x00000024 specifies the size in bytes of the record-specific data that follows.

ObjectTypePen: An EmfPlusPen object that specifies the graphics pen defined by this object record.

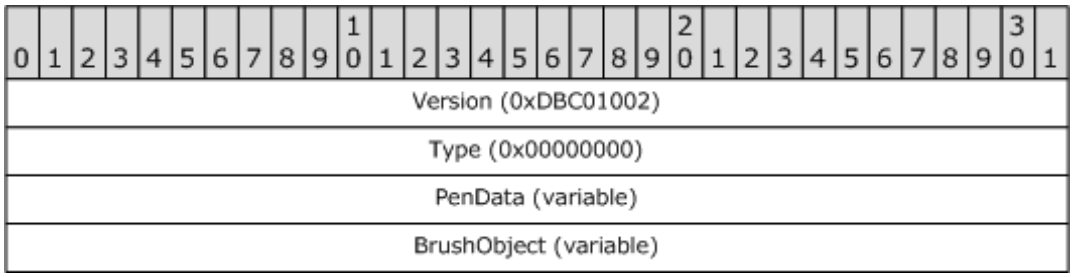


Figure 68: EmfPlusPen Graphics Object Example

Version: 0xDBC01002 specifies the graphics version that was used to create the EmfPlusPen object in this record. The value 0xDBC01002 corresponds to GDI+ version 1.1.

Type: 0x00000000 is not used.

PenData: An EmfPlusPenData object that specifies the characteristics of the graphics pen, including both required and optional fields.

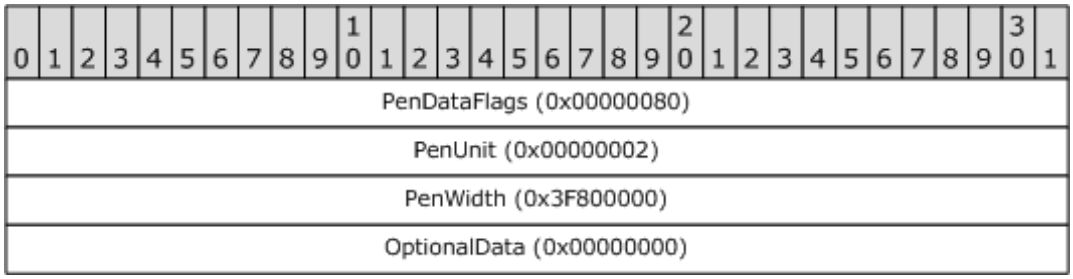


Figure 69: EmfPlusPenData Structure Object Example

PenDataFlags: 0x00000080 specifies that a dashed-line offset value is present in the **OptionalData** field of this object. This is a value from the PenData bit flag constants.

PenUnit: 0x00000002 specifies a unit of one pixel for the pen, from the UnitType enumeration.

PenWidth: 0x3F800000 specifies a floating-point value that is the width of the stroke drawn by the pen, in the units specified by the **PenUnit** field.

OptionalData: 0x00000000 specifies a zero dashed-line offset.

BrushObject: An EmfPlusBrush object that specifies the graphics brush associated with the pen.

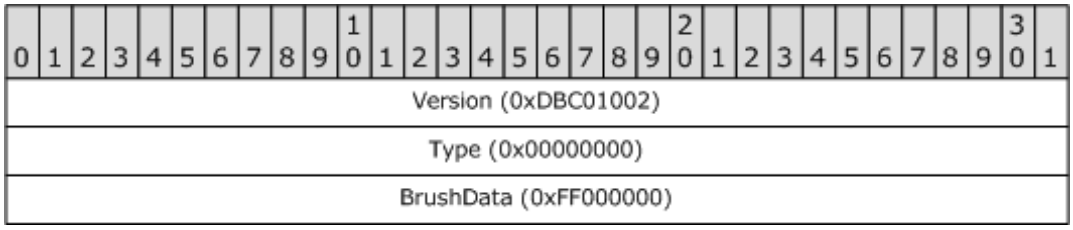


Figure 70: EmfPlusBrush Graphics Object Example

Version: 0xDBC01002 specifies the graphics version that was used to create the EmfPlusBrush object in this record.

Type: 0x00000000 specifies a solid color brush from the BrushType enumeration.

BrushData: 0xFF000000 specifies a blue brush color.

3.2.33.2 EmfPlusDrawPath Example

This section provides an example of the EmfPlusDrawPath record.

```
00000650: 15 40 00 00 10 00 00 00 04 00 00 00 01 00 00 00
```

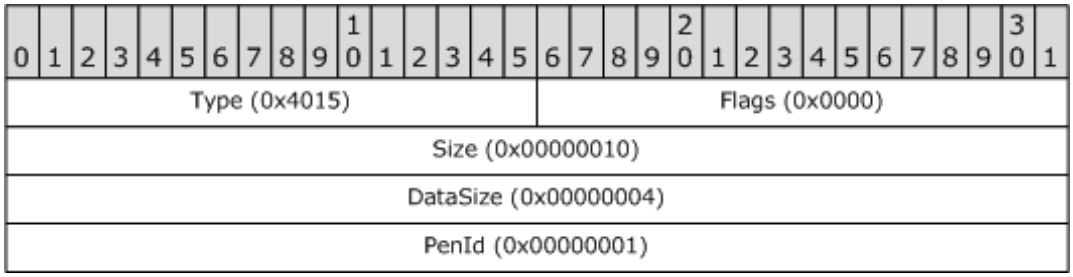


Figure 71: EmfPlusDrawPath Record Example

Type: 0x4015 identifies this record type as EmfPlusDrawPath.

Flags: 0x0000 specifies the graphics object identifier and object type.

Size: 0x00000010 specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

DataSize: 0x00000004 specifies the 32-bit-aligned number of bytes of data in the **RecordData** member that follows. This number does not include the size of the invariant part of this record.

PenId: 0x00000001 specifies the index into the Object Table for the EmfPlusPen object to use.

3.2.34 EMR_RESTOREDC Example 1

This section provides an example of the EMF EMR_RESTOREDC record.

```
00000660: 22 00 00 00 0C 00 00 00 FF FF FF FF
```

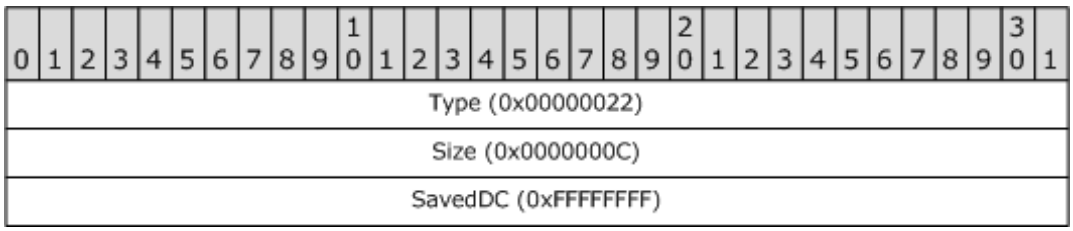


Figure 72: EMF EMR_RESTOREDC Record Example

Type: 0x00000022 identifies the EMF record type as **EMR_RESTOREDC**.

Size: 0x0000000C specifies the size of this record in bytes.

SavedDC: 0xFFFFFFFF specifies the most recently saved state.

3.2.35 EMR_RESTOREDC Example 2

This section provides an example of the EMF EMR_RESTOREDC record.

```
00000660:                           22 00 00 00
00000670: 0C 00 00 00 FF FF FF FF
```

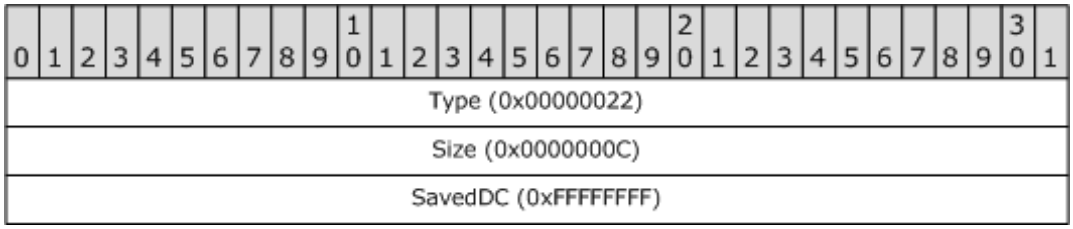


Figure 73: EMF EMR_RESTOREDC Record Example

Type: 0x00000022 identifies the EMF record type as **EMR_RESTOREDC**.

Size: 0x0000000C specifies the size of this record in bytes.

SavedDC: 0xFFFFFFFF specifies the most recently saved state.

3.2.36 EMR_SELECTOBJECT Example 6

This section provides an example of the EMR_SELECTOBJECT record.

```
00000670:                   25 00 00 00 0C 00 00 00
00000680: 0D 00 00 80
```

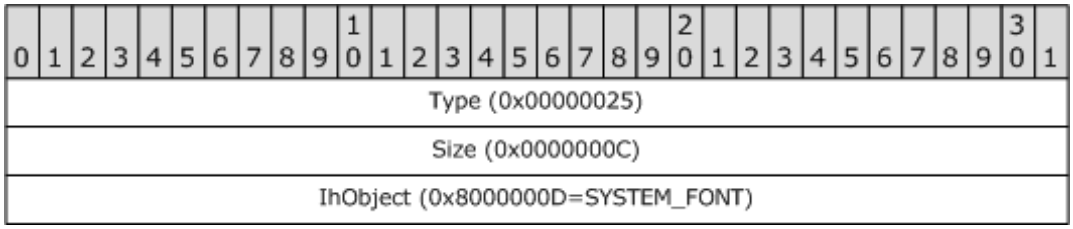


Figure 74: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x8000000D specifies the index of an object in the object table.

3.2.37 EMR_RESTOREDC Example 3

This section provides an example of the EMF EMR_RESTOREDC record.

```
00000680:  22 00 00 00 0C 00 00 00 FF FF FF FF
```

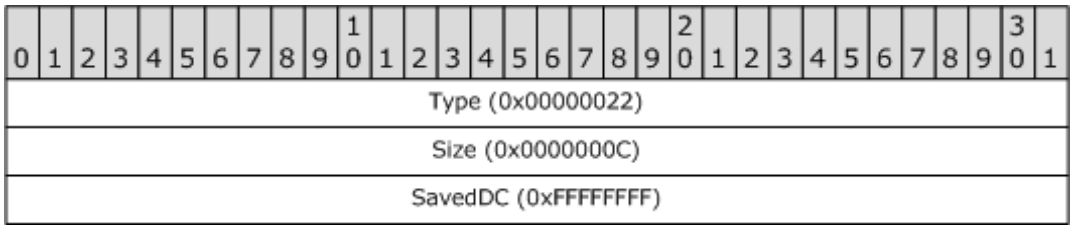


Figure 75: EMF EMR_RESTOREDC Record Example

Type: 0x00000022 identifies the EMF record type as **EMR_RESTOREDC**.

Size: 0x0000000C specifies the size of this record in bytes.

SavedDC: 0xFFFFFFFF specifies the most recently saved state.

3.2.38 EMR_SAVEDC Example 4

This section provides an example of the EMR_SAVEDC record.

```
00000690: 21 00 00 00 08 00 00 00
```

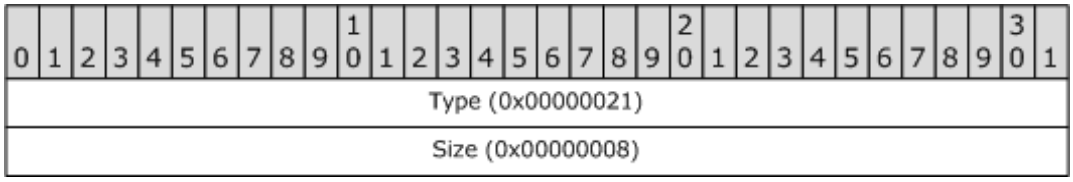


Figure 76: EMF EMR_SAVEDC Record Example

Type: 0x00000021 identifies this EMF record type as EMR_SAVEDC.

Size: 0x00000008 specifies the size of this record in bytes.

3.2.39 EMR_SETLAYOUT Example 3

This section provides an example of the EMF EMR_SETLAYOUT record.

```
00000690:                    73 00 00 00 0C 00 00 00
000006A0: 00 00 00 00
```

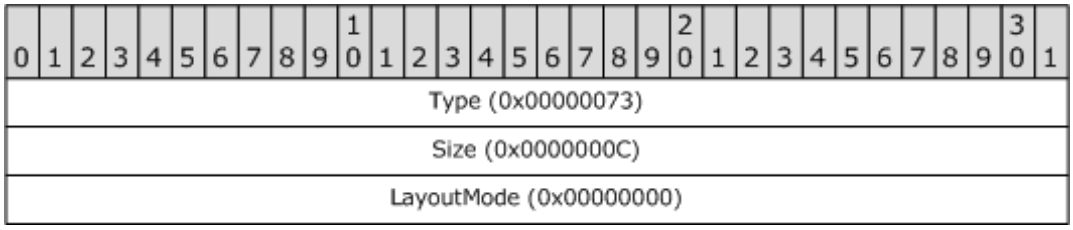


Figure 77: EMF EMR_SETLAYOUT Record Example

Type: 0x00000073 identifies this record type as EMR_SETLAYOUT.

Size: 0x0000000C specifies the size of this record in bytes.

LayoutMode: 0x00000000 specifies left-to-right horizontal layout.

3.2.40 EMR_SETMETARGN Example 2

This section provides an example of the EMR_SETMETARGN record.

```
000006A0:             1C 00 00 00 08 00 00 00
```

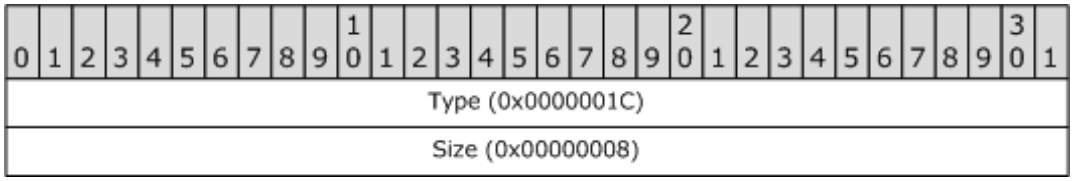


Figure 78: EMF EMR_SETMETARGN Record Example

Type: 0x0000001C identifies this EMF record type as EMR_SETMETARGN.

Size: 0x00000008 specifies the size of this record.

3.2.41 EMR_SELECTOBJECT Example 7

This section provides an example of the EMR_SELECTOBJECT record.

```
000006A0:                         25 00 00 00
000006B0: 0C 00 00 00 00 00 00 80
```

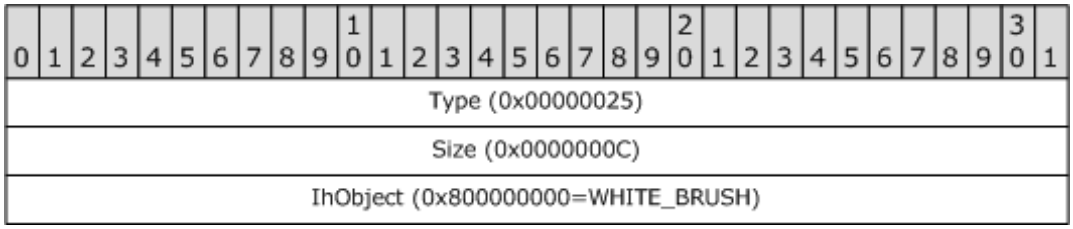


Figure 79: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x80000000 specifies the index of an object in the object table.

3.2.42 EMR_SELECTOBJECT Example 8

This section provides an example of the EMR_SELECTOBJECT record.

```
000006B0:                25 00 00 00 0C 00 00 00
000006C0: 07 00 00 80
```

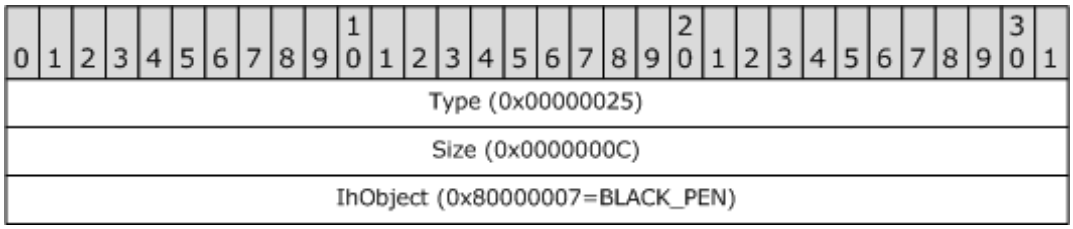


Figure 80: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x80000007 specifies the index of an object in the object table.

3.2.43 EMR_SELECTOBJECT Example 9

This section provides an example of the EMR_SELECTOBJECT record.

```
000006C0: 25 00 00 00 0C 00 00 00 0E 00 00 80
```

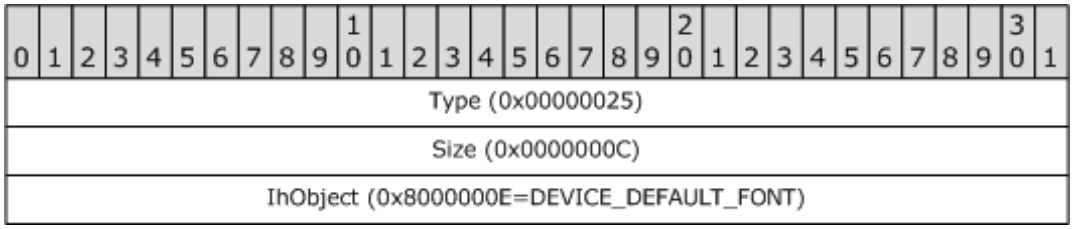


Figure 81: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x8000000E specifies the index of an object in the object table.

3.2.44 EMR_SELECTPALETTE Example 2

This section provides an example of the EMR_SELECTPALETTE record.

```
000006D0: 30 00 00 00 0C 00 00 00 0F 00 00 80
```

Figure 82: EMF EMR_SELECTPALETTE Record Example

Type: 0x00000030 identifies this EMF record type as EMR_SELECTPALETTE.

Size: 0x0000000C specifies the size of this record in bytes.

ihPal: 0x8000000F specifies the palette index. The palette can be selected in background mode only.

3.2.45 EMR_SETBKCOLOR Example 2

This section provides an example of the EMR_SETBKCOLOR record.

```
000006D0:                         19 00 00 00
000006E0: 0C 00 00 00 FF FF FF 00
```

Figure 83: EMF EMR_SETBKCOLOR Record Example

Type: 0x00000019 identifies this EMF record type as EMR_SETBKCOLOR.

Size: 0x0000000C specifies the size of this record in bytes.

Color: 0x00FFFFFF specifies the background color value of the WMF ColorRef object [MS-WMF].

3.2.46 EMR_SETTEXTCOLOR Example 2

This section provides an example of the EMR_SETTEXTCOLOR record.

```
000006E0:                18 00 00 00 0C 00 00 00
000006F0: 00 00 00 00
```

Figure 84: EMF EMR_SETTEXTCOLOR Record Example

Type: 0x00000018 identifies this EMF record type as EMR_SETTEXTCOLOR.

Size: 0x0000000C specifies the size of this record in bytes.

Color: 0x00000000 specifies the text color value.

3.2.47 EMR_SETBKMODE Example 2

This section provides an example of the EMF EMR_SETBKMODE record.

```
000006F0:             12 00 00 00 0C 00 00 00 02 00 00 00
```

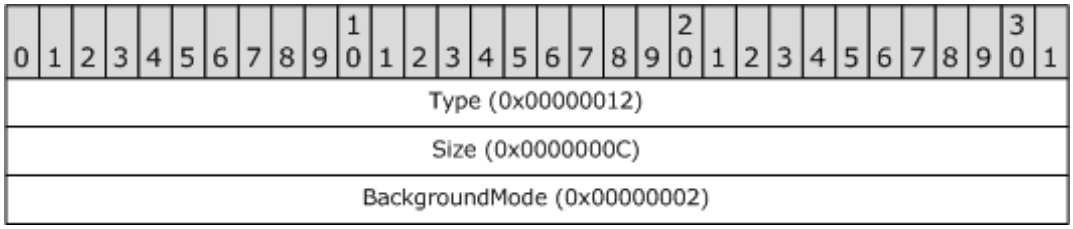


Figure 85: EMF EMR_SETBKMODE Record Example

Type: 0x00000012 identifies the record type as EMR_SETBKMODE.

Size: 0x0000000C specifies the size of this record in bytes.

BackgroundMode: 0x00000002 specifies **OPAQUE** background mode. This value is defined in the EMF BackgroundMode enumeration.

3.2.48 EMR_SETPOLYFILLMODE Example 2

This section provides an example of the EMF EMR_SETPOLYFILLMODE record.

```
00000700: 13 00 00 00 0C 00 00 00 01 00 00 00
```

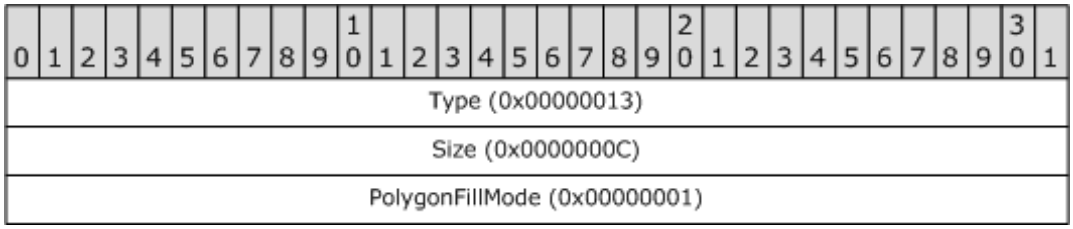


Figure 86: EMF EMR_SETPOLYFILLMODE Record Example

Type: 0x00000013 identifies the record type as EMR_SETPOLYFILLMODE.

Size: 0x0000000C specifies the size of this record in bytes.

PolygonFillMode: 0x00000001 specifies **ALTERNATE** polygon fill mode. This value is defined in the EMF PolygonFillMode enumeration.

3.2.49 EMR_SETROP2 Example 2

This section provides an example of the EMF EMR_SETROP2 record.

```
00000700:                         14 00 00 00
00000710: 0C 00 00 00 0D 00 00 00
```

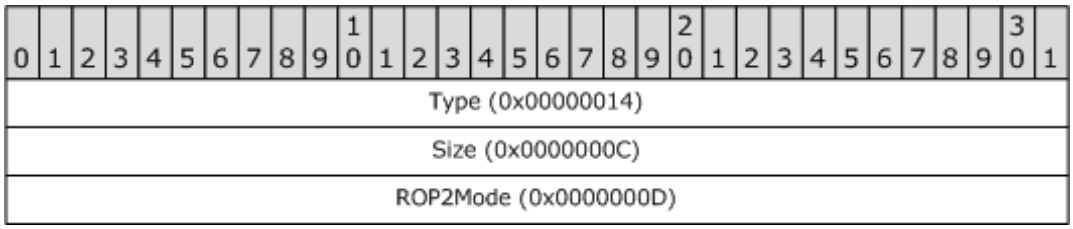


Figure 87: EMF EMR_SETROP2 Record Example

Type: 0x00000014 identifies the record type as EMR_SETROP2.

Size: 0x0000000C specifies the size of this record in bytes.

ROP2Mode: 0x0000000D specifies the **R2_COPYPEN** raster operation mode. This value is defined in the WMF Binary Raster Operation enumeration [MS-WMF].

3.2.50 EMR_SETSTRETCHBLTMODE Example 2

This section provides an example of the EMF EMR_SETSTRETCHBLTMODE record.

```
00000710:                    15 00 00 00 0C 00 00 00
00000720: 01 00 00 00
```

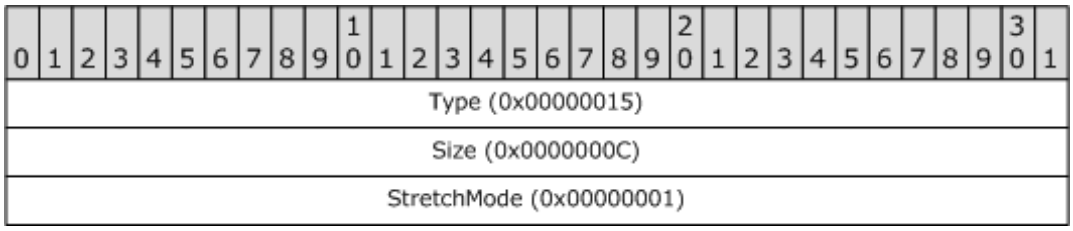


Figure 88: EMF EMR_SETSTRETCHBLTMODE Record Example

Type: 0x00000015 identifies the record type as EMR_SETSTRETCHBLTMODE.

Size: 0x0000000C specifies the size of this record in bytes.

StretchMode: 0x00000001 specifies a Boolean AND operation using the color values for the eliminated and existing pixels, from the EMF StretchMode enumeration.

3.2.51 EMR_SETTEXTALIGN Example 2

This section provides an example of the EMR_SETTEXTALIGN record.

```
00000720:                16 00 00 00 0C 00 00 00 00 00 00 00
```

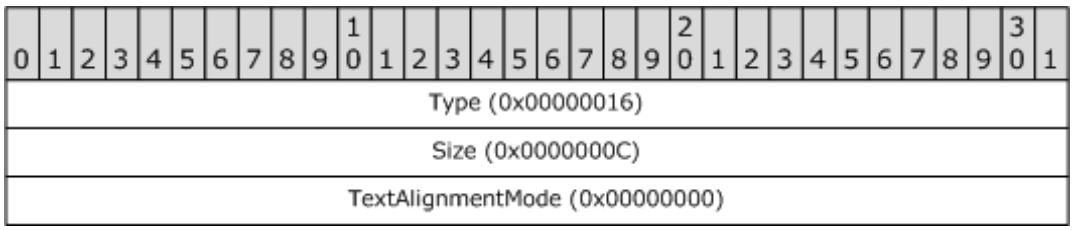


Figure 89: EMF EMR_SETTEXTALIGN Record Example

Type: 0x00000016 identifies this EMF record type as EMR_SETTEXTALIGN.

Size: 0x0000000C specifies the size of this record in bytes.

TextAlignmentMode: 0x00000000 specifies text alignment using a mask of WMF TextAlignmentMode flags or VerticalTextAlignmentMode flags [MS-WMF]. 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.

3.2.52 EMR_SETBRUSHORGEX Example 3

This section provides an example of the EMR_SETBRUSHORGEX record.

```
00000730: 0D 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
```

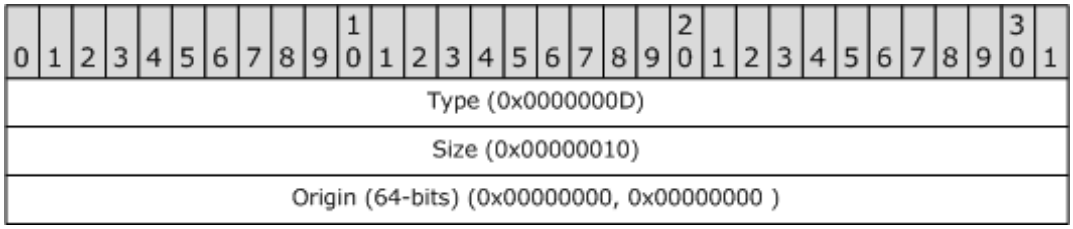


Figure 90: EMF EMR_SETBRUSHORGEX Record Example

Type: 0x0000000D identifies this EMF record type as EMR_SETBRUSHORGEX.

Size: 0x00000010 specifies the size of this record in bytes.

Origin: (0x00000000, 0x00000000) specifies the brush horizontal and vertical origin in device units.

3.2.53 EMR_SETMITERLIMIT Example 2

This section provides an example of the EMF EMR_SETMITERLIMIT record.

```
00000740: 3A 00 00 00 0C 00 00 00 0A 00 00 00
```

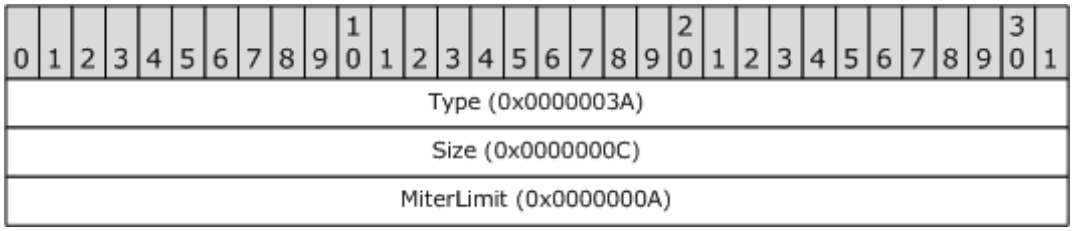


Figure 91: EMF EMR_SETMITERLIMIT Record Example

Type: 0x0000003A identifies the record type as EMR_SETMITERLIMIT.

Size: 0x0000000C specifies the size of this record in bytes.

MiterLimit: 0x0000000A specifies a miter length limit of 10 logical units.

3.2.54 EMR_MOVETOEX Example 2

This section provides an example of the EMR_MOVETOEX record.

```
00000740:                                     1B 00 00 00
00000750: 10 00 00 00 00 00 00 00 00 00 00 00
```

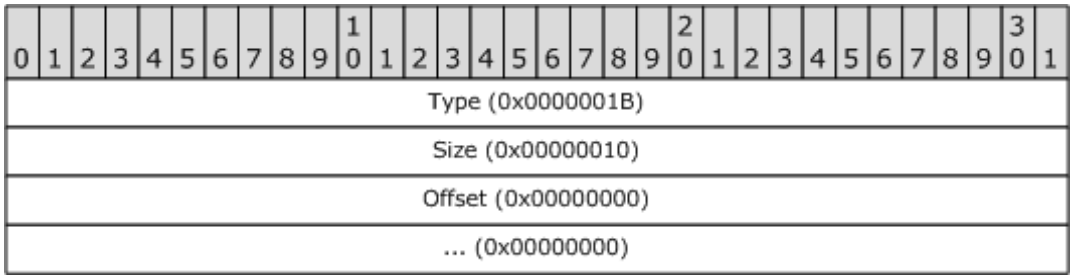


Figure 92: EMF EMR_MOVETOEX Record Example

Type: 0x0000001B identifies this EMF record type as EMR_MOVETOEX.

Size: 0x00000010 specifies the size of this record in bytes.

Offset: (0x00000000, 0x00000000) specifies coordinates of the new current position in logical units.

3.2.55 EMR_SETWORLDTRANSFORM Example 2

This section provides an example of the EMF EMR_SETWORLDTRANSFORM.

```
00000750:                                     23 00 00 00
00000760: 20 00 00 00 87 C3 81 3F 00 00 00 00 00 00 00 00
00000770: 7A BD 80 3F 00 00 00 80 00 00 00 80
```

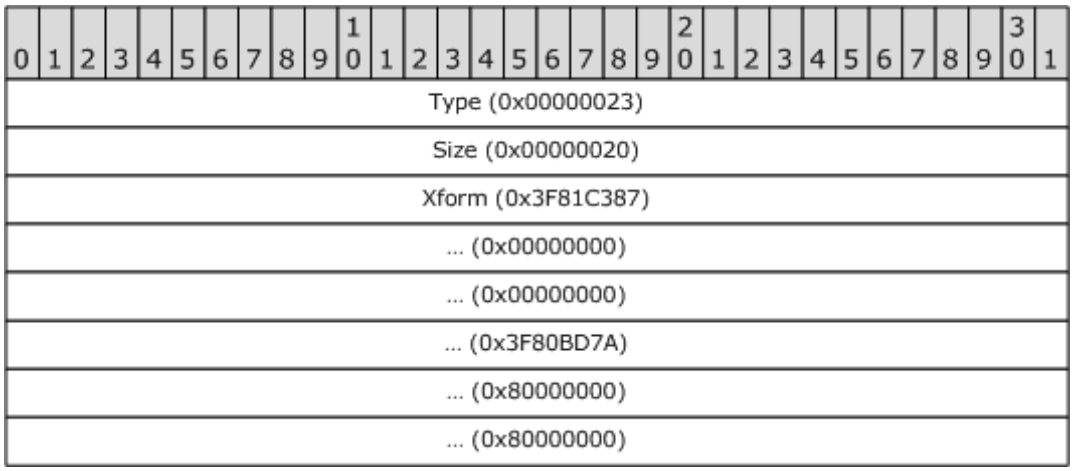


Figure 93: EMF EMR_SETWORLDTRANSFORM Record Example

Type: 0x00000023 identifies the record type as EMR_SETWORLDTRANSFORM.

Size: 0x00000020 specifies the size of this record in bytes.

Xform: (1.005782, 0.000000, 0.000000, 1.013780, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

3.2.56 EMR_MODIFYWORLDTRANSFORM Example 2

This section provides an example of the EMF EMR_MODIFYWORLDTRANSFORM record.

```
00000780: 24 00 00 00 87 C3 81 3F 00 00 00 00 00 00 00 00
00000790: 7A BD 80 3F 00 00 00 80 00 00 00 80 04 00 00 00
```

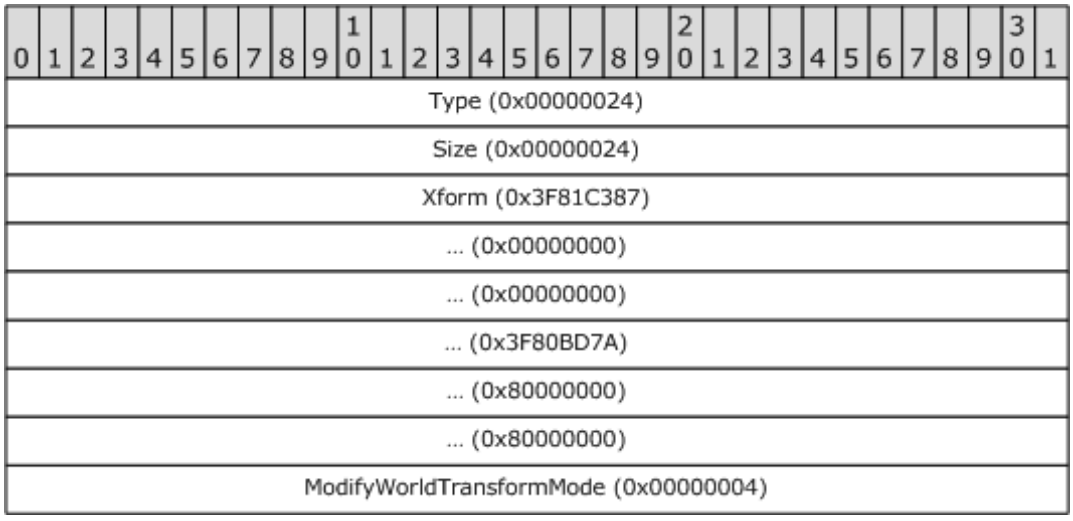


Figure 94: EMF EMR_MODIFYWORLDTRANSFORM Record Example

Type: 0x00000024 identifies the record type as EMR_MODIFYWORLDTRANSFORM.

Size: 0x00000024 specifies the size of this record in bytes.

Xform: (1.005782, 0.000000, 0.000000, 1.013780, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

ModifyWorldTransformMode: 0x00000004 specifies that this record performs the function of an EMF EMR_SETWORLDTRANSFORM. This value is defined in the EMF ModifyWorldTransformMode enumeration.

3.2.57 EMR_SETLAYOUT Example 4

This section provides an example of the EMF EMR_SETLAYOUT record.

```
000007A0: 73 00 00 00 0C 00 00 00 00 00 00 00
```

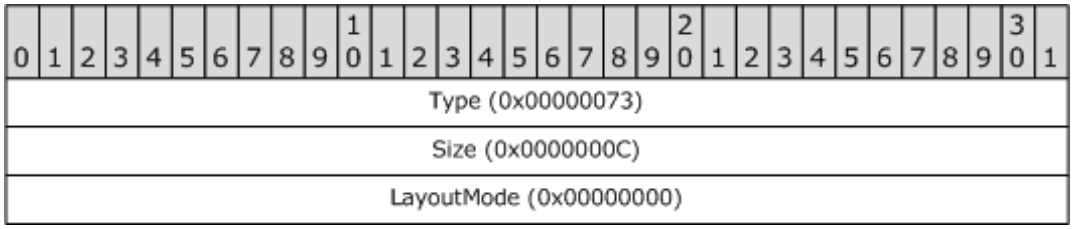


Figure 95: EMF EMR_SETLAYOUT Record Example

Type: 0x00000073 identifies this record type as EMR_SETLAYOUT.

Size: 0x0000000C specifies the size of this record in bytes.

LayoutMode: 0x00000000 specifies left-to-right horizontal layout.

3.2.58 EMR_COMMENT_BEGINGROUP Example

This section provides an example of the EMF EMR_COMMENT_BEGINGROUP record.

```
000007A0:                                       46 00 00 00
000007B0: 60 00 00 00 52 00 00 00 47 44 49 43 02 00 00 00
000007C0: 00 00 00 00 00 00 00 00 66 00 00 00 6C 00 00 00
000007D0: 1B 00 00 00 41 00 70 00 70 00 4E 00 61 00 6D 00
000007E0: 65 00 00 00 49 00 6D 00 61 00 67 00 65 00 20 00
000007F0: 44 00 65 00 73 00 63 00 72 00 69 00 70 00 74 00
00000800: 69 00 6F 00 6E 00 00 00 00 00 00 00
```

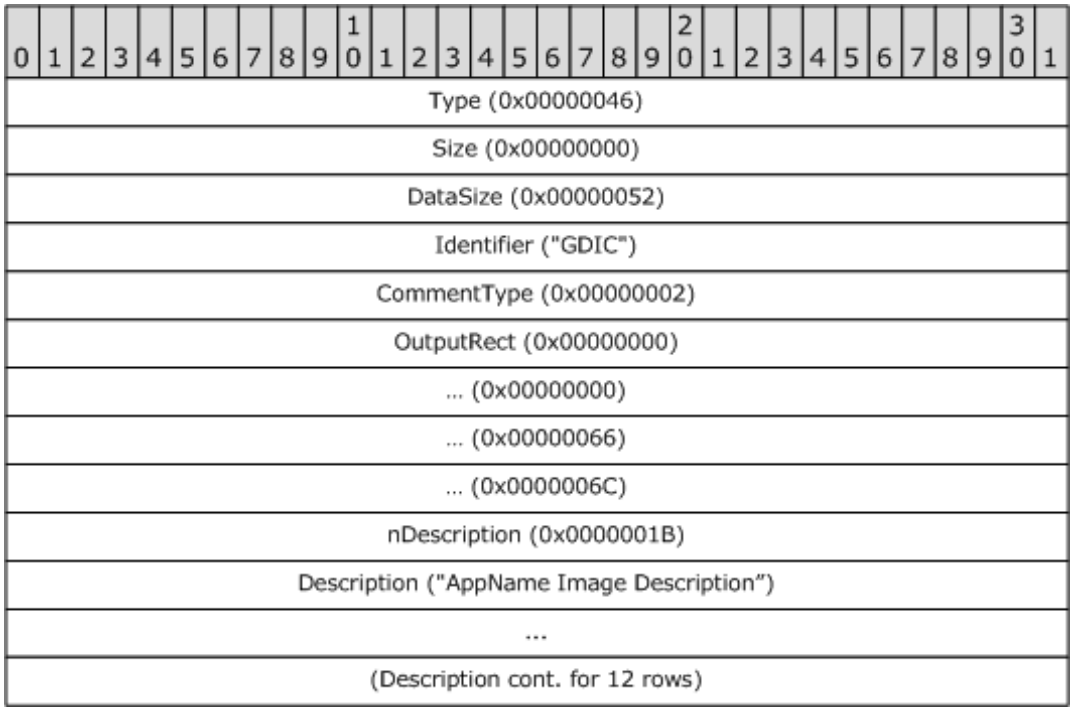


Figure 96: EMF EMR_COMMENT_BEGINGROUP Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x00000060 specifies the size of this record in bytes.

DataSize: 0x00000052 specifies the size of the following fields and data in bytes.

Identifier: "GDIC" identifies this record as an EMF EMR_COMMENT_PUBLIC record.

CommentType: 0x00000002 identifies the type of EMR_COMMENT_PUBLIC record as EMR_COMMENT_BEGINGROUP, from the EMF EmrComment enumeration.

OutputRect: (0x00000000, 0x00000000, 0x00000066, 0x0000006C) defines a WMF RectL object [MS-WMF], which defines the bounding rectangle for output in logical units.

nDescription: 0x0000001B specifies the number of Unicode characters in the description string.

Description: "AppName Image Description".

3.2.59 EMR_SETWORLDTRANSFORM Example 3

This section provides an example of the EMF EMR_SETWORLDTRANSFORM record.

```
00000800:                                       23 00 00 00
00000810: 20 00 00 00 FF FF 7F 3F 00 00 00 00 00 00 00 00
00000820: FD FF 7F 3F 00 00 00 00 00 00 00 00
```

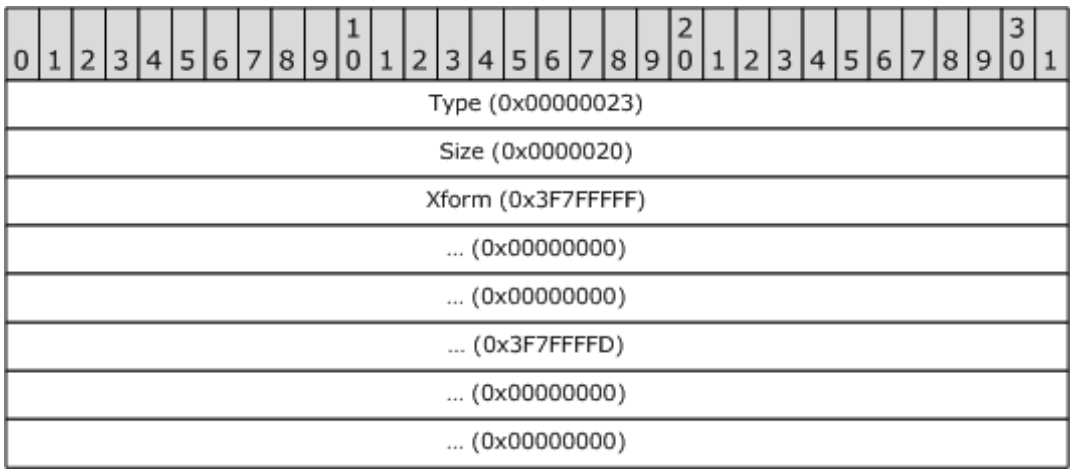


Figure 97: EMF EMR_SETWORLDTRANSFORM Record Example

Type: 0x00000023 identifies the record type as EMR_SETWORLDTRANSFORM.

Size: 0x00000020 specifies the size of this record in bytes.

Xform: (1.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

3.2.60 EMR_MODIFYWORLDTRANSFORM Example 3

This section provides an example of the EMF EMR_MODIFYWORLDTRANSFORM record.

```
00000820:                                       24 00 00 00
00000830: 24 00 00 00 FF FF 7F 3F 00 00 00 00 00 00 00 00
00000840: FD FF 7F 3F 00 00 00 00 00 00 00 00 04 00 00 00
```

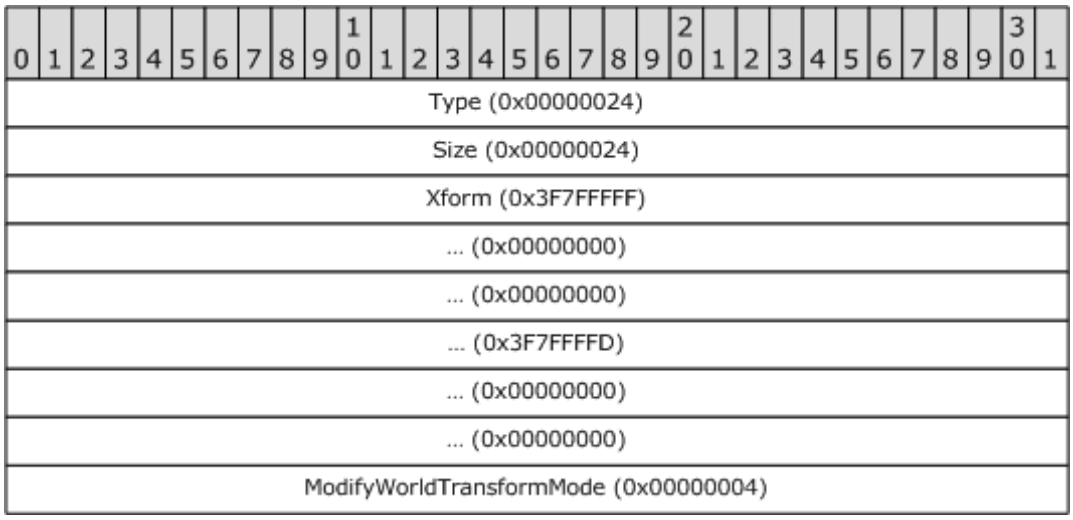


Figure 98: EMF EMR_MODIFYWORLDTRANSFORM Record Example

Type: 0x00000024 identifies the record type as EMR_MODIFYWORLDTRANSFORM.

Size: 0x00000024 specifies the size of this record in bytes.

Xform: (1.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

ModifyWorldTransformMode: 0x00000004 specifies that this record performs the function of an EMF EMR_SETWORLDTRANSFORM record. This value is defined in the EMF ModifyWorldTransformMode enumeration.

3.2.61 EMR_MODIFYWORLDTRANSFORM Example 4

This section provides an example of the EMF EMR_MODIFYWORLDTRANSFORM record.

```
000008B0:             24 00 00 00 24 00 00 00 0D 74 DA 3A
000008C0: 00 00 00 00 00 00 00 00 0C 74 DA 3A 00 00 00 00
000008D0: 00 00 00 00 04 00 00 00
```

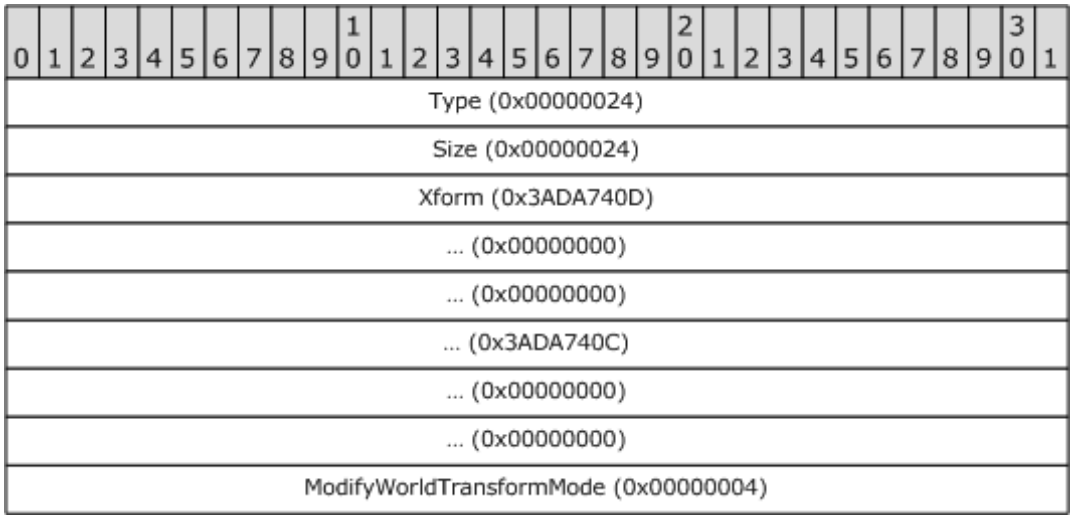


Figure 99: EMF EMR_MODIFYWORLDTRANSFORM Record Example

Type: 0x00000024 identifies the record type as EMR_MODIFYWORLDTRANSFORM.

Size: 0x00000024 specifies the size of this record in bytes.

Xform: (0.001667, 0.000000, 0.000000, 0.001667, 0.000000, 0.000000,) an EMF XForm object, which specifies the world space to page space transformation.

ModifyWorldTransformMode: 0x00000004 specifies that this record performs the function of an EMF EMR_SETWORLDTRANSFORM record. This value is defined in the EMF ModifyWorldTransformMode enumeration.

3.2.62 EMR_MODIFYWORLDTRANSFORM Example 5

This section provides an example of the EMF EMR_MODIFYWORLDTRANSFORM record.

```
000008F0:                         24 00 00 00 24 00 00 00
00000900: 92 5F 2C 3E 00 00 00 00 00 00 00 00 E9 51 38 3E
00000910: 00 00 00 00 00 00 00 00 04 00 00 00
```

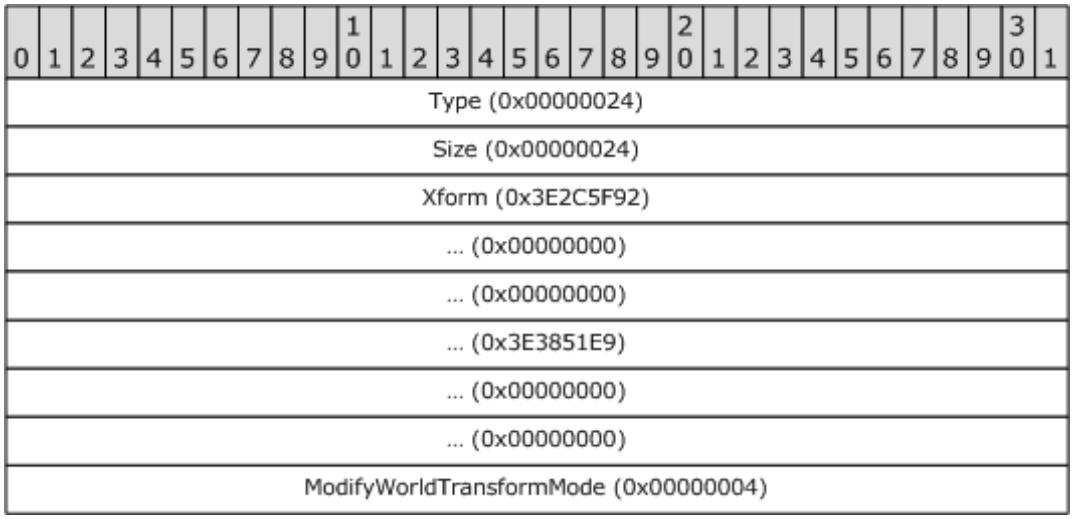


Figure 100: EMF EMR_MODIFYWORLDTRANSFORM Record Example

Type: 0x00000024 identifies the record type as EMR_MODIFYWORLDTRANSFORM.

Size: 0x00000024 specifies the size of this record in bytes.

Xform: (0.180000, 0.000000, 0.000000, 0.168333, 0.000000, 0.000000) an EMF XForm object, which specifies the world space to page space transformation.

ModifyWorldTransformMode: 0x00000004 specifies that this record performs the function of an EMF EMR_SETWORLDTRANSFORM record. This value is defined in the EMF ModifyWorldTransformMode enumeration.

3.2.63 EMR_SELECTOBJECT Example 10

This section provides an example of the EMR_SELECTOBJECT record.

```
00000910:                         25 00 00 00
```

```
00000920: 0C 00 00 00 05 00 00 80
```

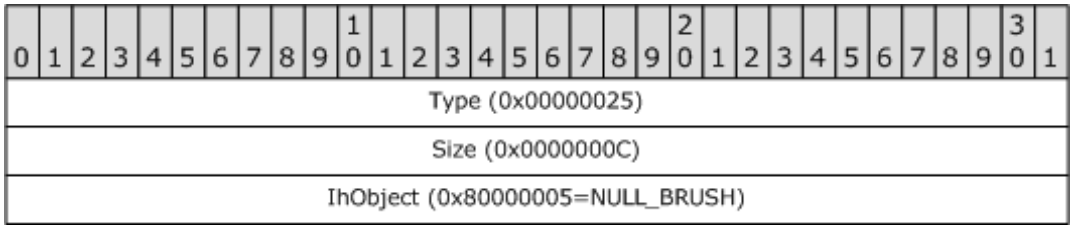


Figure 101: EMF EMR_SELECTOBJECT Record Example

Type: 0x00000025 identifies this EMF record type as EMR_SELECTOBJECT.

Size: 0x0000000C specifies the size of this record in bytes.

ihObject: 0x80000005 specifies the index of an object in the object table.

3.2.64 EMR_ROUNDRECT Example

This section provides an example of the EMR_ROUNDRECT record.

```
00000920:                         2C 00 00 00 20 00 00 00
00000930: FD FF FF FF FE FF FF FF 4F 02 00 00 50 02 00 00
00000940: C8 00 00 00 C8 00 00 00
```

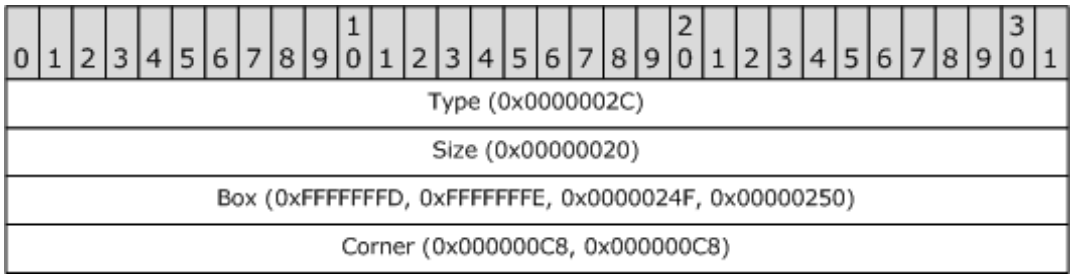


Figure 102: EMF EMR_ROUNDRECT Record Example

Type: 0x0000002C identifies this EMF record type as EMR_ROUNDRECT.

Size: 0x00000020 specifies the size of this record in bytes.

Box: (0xFFFFFFFD, 0xFFFFFFFE, 0x0000024F, 0x00000250) defines the inclusive-inclusive bounding rectangle in logical coordinates.

Corner: (0x000000C8, 0x000000C8) specifies the width and height, in logical coordinates, of the ellipse used to draw the rounded corners.

3.2.65 EMR_COMMENT_ENDGROUP Example

This section provides an example of the EMF EMR_COMMENT_ENDGROUP record.

```
00000940:                               46 00 00 00 14 00 00 00
00000950: 08 00 00 00 47 44 49 43 03 00 00 00
```

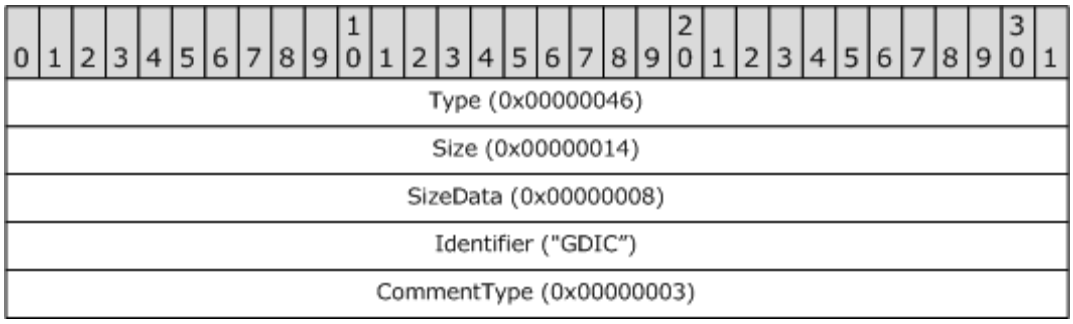


Figure 103: EMF EMR_COMMENT_ENDGROUP Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x00000014 specifies the size of this record in bytes.

DataSize: 0x00000008 specifies the size of the following fields and data in bytes.

Identifier: "GDIC" identifies this record as an EMR_COMMENT_PUBLIC record.

CommentType: 0x00000003 identifies the type of EMR_COMMENT_PUBLIC record as EMR_COMMENT_ENDGROUP, from the EMF EmrComment enumeration.

3.2.66 EMR_RESTOREDC Example 4

This section provides an example of the EMF EMR_RESTOREDC record.

```
00000950:                            22 00 00 00
00000960: 0C 00 00 00 FF FF FF FF
```

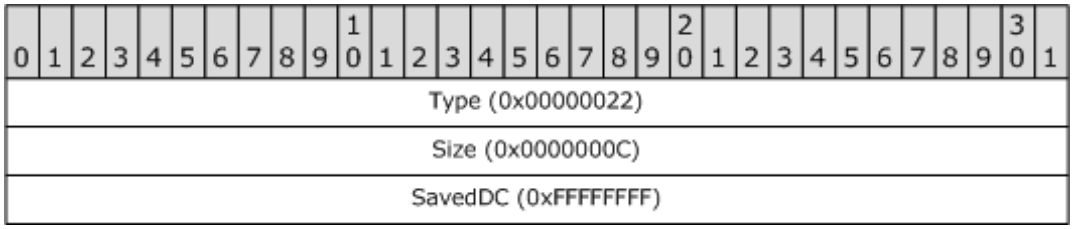


Figure 104: EMF EMR_RESTOREDC Record Example

Type: 0x00000022 identifies the EMF record type as **EMR_RESTOREDC**.

Size: 0x0000000C specifies the size of this record in bytes.

SavedDC: 0xFFFFFFFF specifies the most recently saved state.

3.2.67 EMR_COMMENT_EMFPLUS Example 4

This section provides an example of the EMF EMR_COMMENT_EMFPLUS record.

```
00000960:                      46 00 00 00 54 00 00 00
00000970: 48 00 00 00 45 4D 46 2B
```

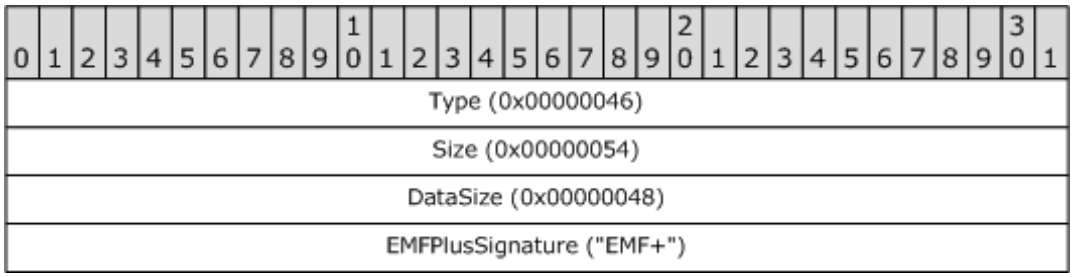


Figure 105: EMF EMR_COMMENT_EMFPLUS Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x00000054 specifies the size of this record in bytes, including embedded EMF+ records.

DataSize: 0x00000048 specifies the size of the EMF+ data in bytes.

EMFPlusSignature: "EMF+" identifies the comment record type as EMR_COMMENT_EMFPLUS.

The embedded EMF+ records are presented in the sections that follow.

3.2.67.1 EmfPlusEndContainer Example

This section provides an example of the EmfPlusEndContainer record.

```
00000970:                         40 00 00 10 00 00 00
00000980: 04 00 00 00 01 00 00 00
```

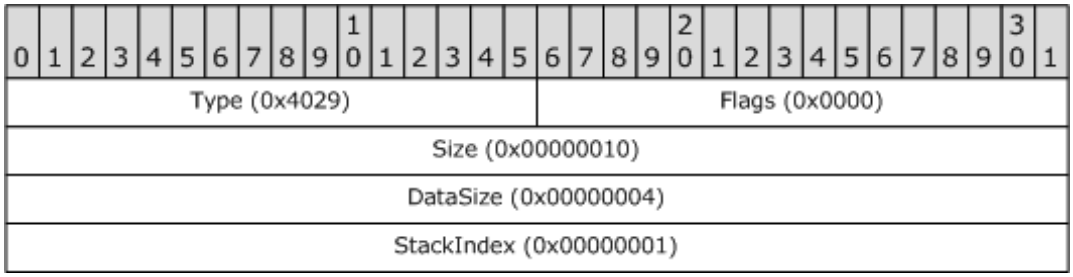


Figure 106: EmfPlusEndContainer Record Example

Type: 0x4029 identifies this record type as EmfPlusEndContainer.

Flags: 0x0000 This field is undefined for this record type.

Size: 0x00000010 specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

DataSize: 0x00000004 specifies the 32-bit-aligned number of bytes of data in the **PointData** member that follows. This number does not include the size of the invariant part of this record.

StackIndex: 0x00000001 specifies the identifier used to reference the container in future records.

3.2.67.2 EmfPlusRestore Example

This section provides an example of the EmfPlusRestore record.

```
00000980:                         26 40 00 00 10 00 00 00
00000990: 04 00 00 00 00 00 00 00
```

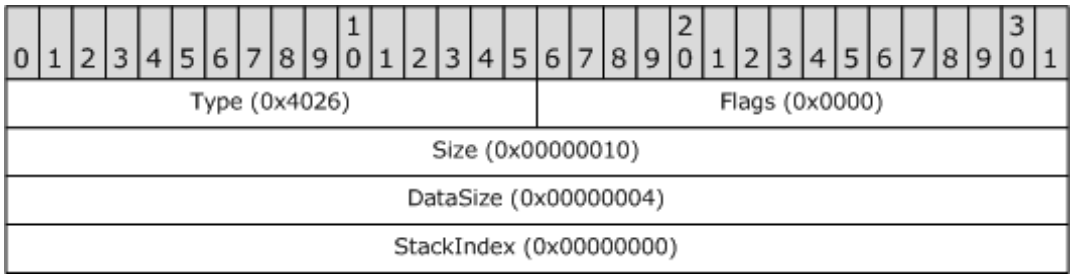


Figure 107: EmfPlusRestore Record Example

Type: 0x4026 identifies this record type as EmfPlusRestore.

Flags: 0x0000 This value is undefined for this record type.

Size: 0x00000010 specifies the 32-bit-aligned size of the entire record in bytes, including the 12-byte record header and the record-specific buffer data.

DataSize: 0x00000004 specifies the 32-bit-aligned number of bytes of data in the **PointData** member that follows. This number does not include the size of the invariant part of this record.

StackIndex: 0x00000000 specifies the identifier used to retrieve the graphics state from the correct save level on the graphics state stack.

3.2.67.3 EmfPlusSetWorldTransform Example

This section provides an example of the EmfPlusSetWorldTransform record.

```
00000990:                         2A 40 00 00 24 00 00 00
000009A0: 18 00 00 00 00 00 80 3F 00 00 00 00 00 00 00 00
000009B0: 00 00 80 3F 00 00 00 00 00 00 00 00
```

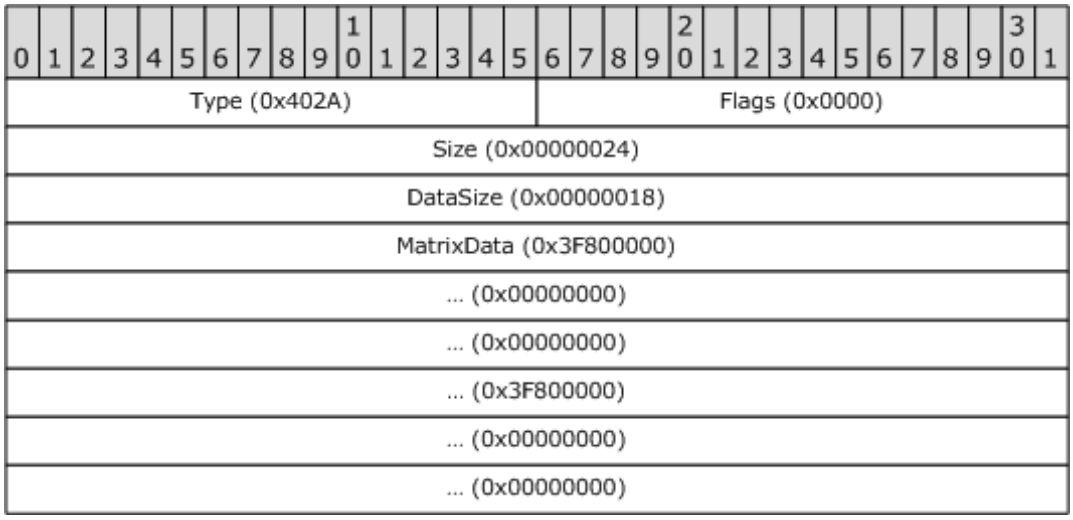


Figure 108: EmfPlusSetWorldTransform Record Example

Type: 0x402A identifies the record type as EmfPlusSetWorldTransform.

Flags: 0x0000 is undefined for this record type.

Size: 0x00000024 specifies the size in bytes of the entire record.

DataSize: 0x00000018 specifies the size in bytes of the **MatrixData** field that follows.

MatrixData: An EmfPlusTransformMatrix object that contains the world space transform.

3.2.68 EMR_BITBLT Example

This section provides an example of the EMF EMR_BITBLT record.

```
000009B0:                                       4C 00 00 00
000009C0: 64 00 00 00 FF FF FF FF FF FF FF FF 64 00 00 00
000009D0: 6B 00 00 00 FF FF FF FF FF FF FF FF 66 00 00 00
000009E0: 6D 00 00 00 29 00 AA 00 00 00 00 00 00 00 00 00
000009F0: 00 00 80 3F 00 00 00 00 00 00 00 00 00 00 80 3F
00000A00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000A10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

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
Type (0x0000004C)																															
Size (0x00000064)																															
Bounds (128-bits) (0xFFFFFFFF, 0xFFFFFFFF, 0x00000064, 0x0000006B)																															
xDest (0xFFFFFFFF)																															
yDest (0xFFFFFFFF)																															
cxDest (0x00000066)																															
cyDest (0x0000006D)																															
BitBlitRasterOperation (0x00AA0029)																															
xSrc (0x00000000)																															
ySrc (0x00000000)																															
xformSrc (0x3F800000, 0x00000000, 0x00000000, 0x3F800000, 0x00000000, 0x00000000)																															
BkColorSrc (0x00000000)																															

Figure 109: EMF EMR_BITBLT Record Example, Part 1

Type: 0x0000004C identifies the record type as EMR_BITBLT.

Size: 0x00000064 specifies the size of this record in bytes.

Bounds: (0xFFFFFFFF, 0xFFFFFFFF, 0x00000064, 0x0000006B) defines the bounding rectangle in device units.

xDest: 0xFFFFFFFF specifies the logical x-coordinate of the upper-left corner of the destination rectangle.

yDest: 0xFFFFFFFF specifies the logical y-coordinate of the upper-left corner of the destination rectangle.

cxDest: 0x00000066 specifies the logical width of the destination rectangle.

cyDest: 0x0000006D specifies the logical height of the destination rectangle.

BitBlitRasterOperation: 0x00AA0029 specifies the **raster operation** code. These codes define how the color data of the source rectangle is to be combined with the color data of the destination rectangle to achieve the final color. This value is defined in the WMF Ternary Raster Operation enumeration [MS-WMF].

xSrc: 0x00000000 specifies the logical x-coordinate of the upper-left corner of the source rectangle.

ySrc: 0x00000000 specifies the logical y-coordinate of the upper-left corner of the source rectangle.

xformSrc: (0x3F800000, 0x00000000, 0x00000000, 0x3F800000, 0x00000000, 0x00000000) defines the world space to page space transformation of the source bitmap.

BkColorSrc: 0x00000000 specifies the background color of the source bitmap.

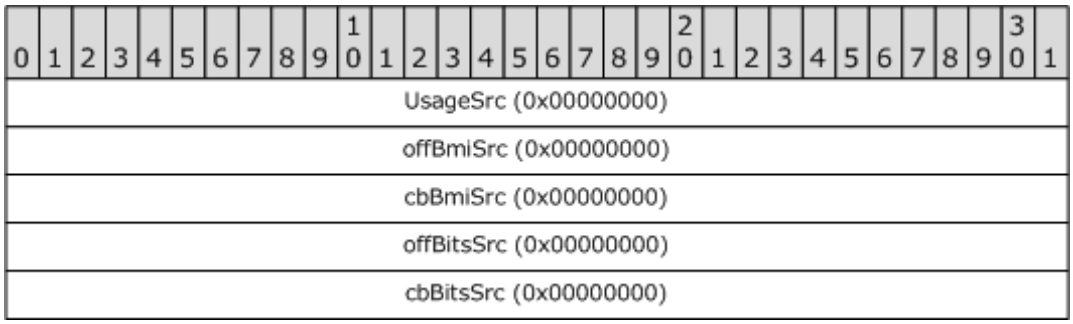


Figure 110: EMF EMR_BITBLT Record Example, Part 2

UsageSrc: 0x00000000 specifies that the color table in the WMF DeviceIndependentBitmap object [MS-WMF] header contains RGB values. This value is defined in the EMF DIBColors enumeration.

offBmiSrc: 0x00000000 specifies the offset to the source **device-independent bitmap (DIB)** object header.

cbBmiSrc: 0x00000000 specifies the size of the source DIB object header.

offBitsSrc: 0x00000000 specifies the offset to the source bitmap bits.

cbBitsSrc: 0x00000000 specifies the size of the source bitmap bits.

3.2.69 EMR_RESTOREDC Example 5

This section provides an example of the EMF EMR_RESTOREDC record.

```
00000A20: 22 00 00 00 0C 00 00 00 FF FF FF FF
```

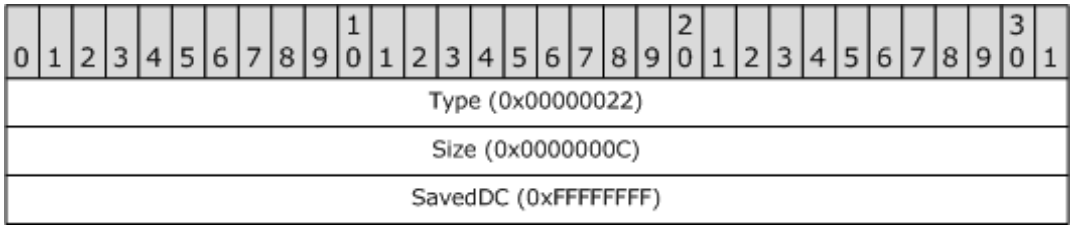


Figure 111: EMF EMR_RESTOREDC Record Example

Type: 0x00000022 identifies the EMF record type as **EMR_RESTOREDC**.

Size: 0x0000000C specifies the size of this record in bytes.

SavedDC: 0xFFFFFFFF specifies the most recently saved state.

3.2.70 EMR_COMMENT_EMFPLUS Example 5

This section provides an example of the EMF EMR_COMMENT_EMFPLUS record.

```
00000A20:                                       46 00 00 00
00000A30: 1C 00 00 00 10 00 00 00 45 4D 46 2B
```

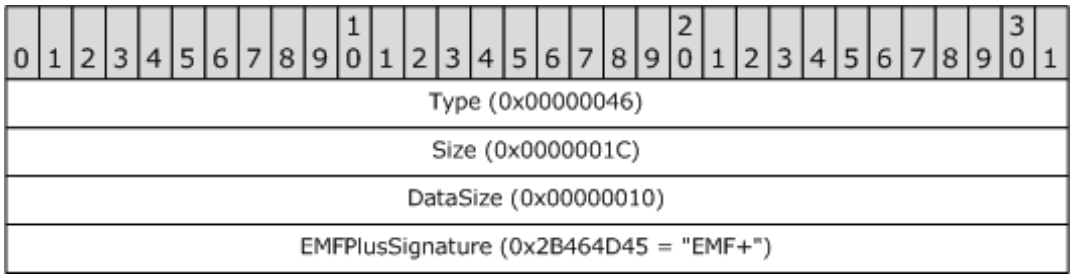


Figure 112: EMF EMR_COMMENT_EMFPLUS Record Example

Type: 0x00000046 identifies this record as an EMF comment record.

Size: 0x0000001C specifies the size of this record in bytes, including embedded EMF+ records.

DataSize: 0x00000010 specifies the size of the EMF+ data in bytes.

EMFPlusSignature: "EMF+" identifies the comment record type as EMR_COMMENT_EMFPLUS.

The embedded EMF+ records are presented in the sections that follow.

3.2.70.1 EmfPlusEndOfFile Example

This section provides an example of the EmfPlusEndOfFile record.

```
00000A30:                                       02 40 00 00
00000A40: 0C 00 00 00 00 00 00 00
```

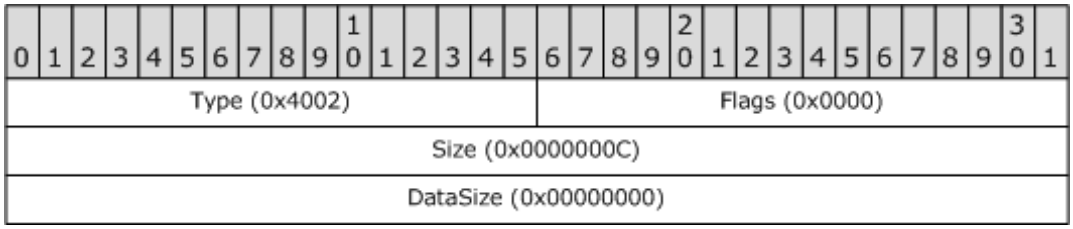


Figure 113: EmfPlusEndOfFile Record Example

Type: 0x4002 identifies this record type as EmfPlusEndOfFile.

Flags: 0x0000 is not used by this record type.

Size: 0x0000000C specifies the 32-bit-aligned size of this record in bytes.

DataSize: 0x00000000 specifies that no bytes of data follow.

3.2.71 EMR_EOF Example

This section provides an example of the EMF EMR_EOF record.

```
00000A40:                   0E 00 00 00 14 00 00 00
00000A50: 00 00 00 00 10 00 00 00 14 00 00 00
```

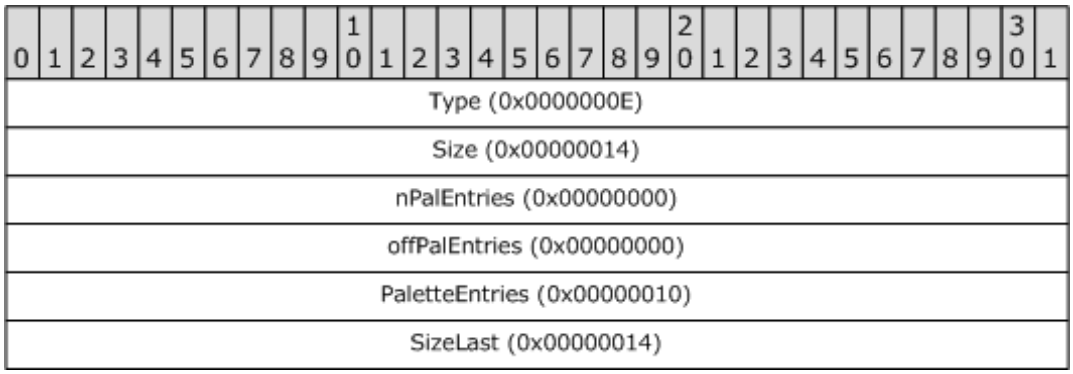


Figure 114: EMF EMR_EOF Record Example

Type: 0x0000000E identifies the record type as EMR_EOF.

Size: 0x00000014 specifies the size of this record in bytes.

nPalEntries: 0x00000000 specifies the number of palette entries.

offPalEntries: 0x00000000 is ignored, as no palette entries are defined.

PaletteEntries: 0x00000010 is ignored, as no palette entries are defined.

SizeLast: 0x00000014 specifies a value that is the same as **Size**, and is the last field of the metafile.

3.3 EMF+ String Drawing Example

This section provides an example of EMF+ string drawing as generated by GDI+ functions.

The following GDI+ example generates an EmfPlusDrawString record and EmfPlusFont and EmfPlusStringFormatData objects.

```
GdiplusStartupInput gdiplusStartupInput;
GdiplusStartupOutput gdiplusStartupOutput;
ULONG PTR gdiplusToken = 0;

int CALLBACK WinMain( HINSTANCE, HINSTANCE, LPSTR, int )
{
    // InitializeGdiPlus
```

```
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, &gdiplusStartupOutput);

    HDC DeviceContext = GetDC( nullptr );

    Metafile File( L"DrawString.emf", DeviceContext, EmfTypeEmfPlusOnly, L"DrawString Demo"
);
    Graphics RenderTarget( &File );
    FontFamily Family( L"Arial" );
    Font EffectiveFont( &Family, 40.0f, FontStyle::FontStyleUnderline, UnitPixel );
    RectF LayoutRect( 0, 0, 100, 100 );

    Gdiplus::StringFormat Format( 0, 0 );
    Format.SetAlignment( Gdiplus::StringAlignment::StringAlignmentFar );

    Gdiplus::Rect GradientRect( 0, 0, 100, 100 );

    Gdiplus::LinearGradientBrush
        TestBrush( GradientRect,
                   Gdiplus::Color( 0xff, 0x00, 0x00 ),
                   Gdiplus::Color( 0x00, 0x00, 0xff ),
                   0.0f );

    const wchar t HelloWorld[] = L"Hello World 1 2 3 4!";

    RenderTarget.DrawString( HelloWorld,
                             ARRAYSIZE(HelloWorld) - 1,
                             &EffectiveFont,
                             LayoutRect,
                             &Format,
                             &TestBrush );

    ReleaseDC( nullptr, DeviceContext );

    return 1;
}
```

The EMF+ metafile generated by the preceding GDI+ example renders the following image:

Figure 115: EMF+ string drawing example

4 Security

This file format enables third parties to send payloads, such as PostScript, to pass through as executable code.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system
- Windows Server 2016 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.3.1: The following table shows how the two cases of embedded EMF+ records are processed on different versions of Windows.

Metafile type	Processed by	Case I	Case II
EMF+ only	Windows NT 3.1 operating system, Windows NT 3.5 operating system, Windows NT 3.51 operating system, and Windows NT 4.0 operating system	No records are processed.	No records are processed.
EMF+ only	Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2 operating system, Windows 8, Windows Server 2012 operating system, Windows 8.1, Windows Server 2012 R2 operating system, and Windows 10	EMF records 3 through N are ignored; EMF+ records are processed.	EMF records 3 through N are ignored; EMF+ records are processed.
EMF+ dual	Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0	EMF records 3 through N are processed; EMF+	EMF records 3 through N are processed; EMF+

Metafile type	Processed by	Case I	Case II
		records are ignored.	records are ignored.
EMF+ dual	Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, and Windows 10	EMF records 3 through N are ignored; EMF+ records are processed.	EMF records 3 through N are processed; EMF+ records are processed.

<2> Section 2.1.1.6: Windows treats any invalid values as CompositingQualityDefault.

<3> Section 2.1.1.12: GDI+ version 1.1 is not supported on Windows 2000.

<4> Section 2.1.1.28: **SmoothingModeAntiAlias8x4** is not supported on Windows 2000.

<5> Section 2.1.1.28: **SmoothingModeAntiAlias8x8** is not supported on Windows 2000.

<6> Section 2.1.2.1: Windows-based GDI+ ignores blend factors for BrushDataBlendFactorsV.

<7> Section 2.1.2.3: Windows does not write this flag value to EMF+ metafiles.

<8> Section 2.1.2.8: In Windows, this flag is set to specify that GDI+ is used to render text instead of GDI.

<9> Section 2.1.3.1: Windows produces corrupt records when the ColorCurve effect is used.

<10> Section 2.2.1.3: Windows never emits unit type **UnitTypeDisplay** for this field.

<11> Section 2.2.2.19: If the graphics version number is **GraphicsVersion1**, the metafile was created using Windows GDI+ 1.0.

If the graphics version number is **GraphicsVersion1_1**, the metafile was created using Windows GDI+ 1.1.

<12> Section 2.2.2.24: When **Reserved1** is not the same as **StartColor** and **Reserved2** is not the same as **EndColor**, nothing will be rendered.

<13> Section 2.2.2.24: When **Reserved1** is not the same as **StartColor** and **Reserved2** is not the same as **EndColor**, nothing will be rendered.

<14> Section 2.2.2.25: In Windows-based GDI+, the result of BrushDataBlendFactorsV alone is the same as if no blend factor was used.

<15> Section 2.2.2.25: In Windows-based GDI+, when both BrushDataBlendFactorsV and BrushDataBlendFactorsH are set, the result is the same as if BrushDataBlendFactorsH alone is set.

<16> Section 2.2.3: Image Effects Objects are only supported by GDI+ 1.1.

<17> Section 2.2.3.6: The client treats this value as 1.0.

<18> Section 2.3.2.1: Windows sets this field to an arbitrary value between 0x0000 and 0x0003, inclusive.

<19> Section 2.3.4.3: In Windows, if the **P** flag is set, the **C** flag is clear.

<20> Section 2.3.4.4: In Windows, if the **P** flag is set, the **C** flag is clear.

<21> Section 2.3.4.9: In Windows, if the **P** flag is set, the **C** flag is clear.

<22> Section 2.3.4.10: In Windows, if the **P** flag is set, the **C** flag is clear.

<23> Section 2.3.4.15: In Windows, if the **P** flag is set, the **C** flag is clear.

<24> Section 2.3.4.19: In Windows, if the **P** flag is set, the **C** flag is clear.

<25> Section 2.3.5.2: The EmfPlusSerializableObject record type is only supported by GDI+ 1.1.

<26> Section 2.3.6.1: Smoothing with anti-aliasing that uses a box filter algorithm is not supported on Windows 2000.

<27> Section 2.3.7.1: Windows never writes those values to the **PageUnit** field, but they are accepted with undefined results.

<28> Section 2.3.8: Windows does not generate the EmfPlusSetTSClip record.

<29> Section 2.3.8: Windows does not generate the EmfPlusSetTSGraphics record.

<30> Section 2.3.8.1: The EmfPlusSetTSClip record type is only supported by GDI+ 1.1.

<31> Section 2.3.8.2: The EmfPlusSetTSGraphics record type is only supported by GDI+ 1.1.

<32> Section 2.3.9.5: Windows never writes those values to the **PageUnit** field, but they are accepted with undefined results.

6 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.3.6 ColorMatrixEffect Object	Removed "MUST" from product behavior note.	N	Content update.
2.3.7.3 EmfPlusEndContainer Record	Changed "must" to "MUST".	N	Content update.
2.3.9.1 EmfPlusMultiplyWorldTransform Record	Changed "should" to "is".	N	Content update.
2.3.9.3 EmfPlusRotateWorldTransform Record	Changed "should" to "is".	N	Content update.
2.3.9.4 EmfPlusScaleWorldTransform Record	Changed "should" to "is".	N	Content update.
2.3.9.7 EmfPlusTranslateWorldTransform Record	Changed "should" to "is".	N	Content update.

7 Index

A

Applicability 21

B

Bit flags constant types 60
BitmapDataType enumeration 28
BlurEffect Object 109
BlurEffect packet 109
BlurEffectGuid 66
BrightnessContrastEffect Object 110
BrightnessContrastEffect packet 110
BrightnessContrastEffectGuid 66
BrushDataBlendFactorsH 61
BrushDataBlendFactorsV 61
BrushDataDoNotTransform 61
BrushDataFocusScales 61
BrushDataIsGammaCorrected 61
BrushDataPath 61
BrushDataPresetColors 61
BrushDataTransform 61
BrushType enumeration 29
Byte ordering 20
Byte ordering example 193

C

Change tracking 260
Clipping_Record_Types packet 121
ColorBalanceEffect Object 111
ColorBalanceEffect packet 111
ColorBalanceEffectGuid 66
ColorCurveEffect Object 112
ColorCurveEffect packet 112
ColorCurveEffectGuid 66
ColorLookupTableEffect Object 113
ColorLookupTableEffect packet 113
ColorLookupTableEffectGuid 66
ColorMatrixEffect Object 114
ColorMatrixEffect packet 114
ColorMatrixEffectGuid 66
CombineMode enumeration 29
Common data types and fields 23
CompositingMode enumeration 30
CompositingQuality enumeration 30
Constants 23
Constants - EMF+ 23
Control_Record_Types packet 127
CurveAdjustments enumeration 31
CurveChannel enumeration 32
CustomLineCapDataFillPath 62
CustomLineCapDataLinePath 62
CustomLineCapDataType enumeration 32

D

DashedLineCapType enumeration 32
Data types and fields - common 23
Details
  common data types and fields 23
Drawing_Record_Types packet 130
DriverStringOptionsCmapLookup 62
DriverStringOptionsLimitSubpixel 62
DriverStringOptionsRealizedAdvance 62
DriverStringOptionsVertical 62

E

EMF+ Metafile Example example 194
EMF+ metafile examples 194
EMF+ String Drawing Example example 254
EmfPlusARGB Object 77
EmfPlusARGB packet 77
EmfPlusBeginContainer packet 177
EmfPlusBeginContainer Record 177
EmfPlusBeginContainerNoParams example 221
EmfPlusBeginContainerNoParams packet 179
EmfPlusBeginContainerNoParams Record 179
EmfPlusBitmap Object 78
EmfPlusBitmap packet 78
EmfPlusBitmapData Object 79
EmfPlusBitmapData packet 79
EmfPlusBlendColors Object 80
EmfPlusBlendColors packet 80
EmfPlusBlendFactors Object 81
EmfPlusBlendFactors packet 81
EmfPlusBoundaryPathData Object 82
EmfPlusBoundaryPathData packet 82
EmfPlusBoundaryPointData Object 82
EmfPlusBoundaryPointData packet 82
EmfPlusBrush Object 67
EmfPlusBrush packet 67
EmfPlusCharacterRange Object 83
EmfPlusCharacterRange packet 83
EmfPlusClear packet 132
EmfPlusClear Record 132
EmfPlusComment packet 126
EmfPlusComment Record 126
EmfPlusCompoundLineData Object 83
EmfPlusCompoundLineData packet 83
EmfPlusCompressedImage Object 84
EmfPlusCompressedImage packet 84
EmfPlusCustomEndCapData Object 84
EmfPlusCustomEndCapData packet 84
EmfPlusCustomLineCap Object 68
EmfPlusCustomLineCap packet 68
EmfPlusCustomLineCapArrowData Object 85
EmfPlusCustomLineCapArrowData packet 85
EmfPlusCustomLineCapData Object 86
EmfPlusCustomLineCapData packet 86
EmfPlusCustomLineCapOptionalData Object 87
EmfPlusCustomLineCapOptionalData packet 87
EmfPlusCustomStartCapData Object 88
EmfPlusCustomStartCapData packet 88
EmfPlusDashedLineData Object 88
EmfPlusDashedLineData packet 88
EmfPlusDrawArc packet 133
EmfPlusDrawArc Record 133
EmfPlusDrawBeziers packet 134
EmfPlusDrawBeziers Record 134
EmfPlusDrawClosedCurve packet 136

EmfPlusDrawClosedCurve Record 136
EmfPlusDrawCurve packet 139
EmfPlusDrawCurve Record 139
EmfPlusDrawDriverString packet 141
EmfPlusDrawDriverString Record 141
EmfPlusDrawEllipse packet 143
EmfPlusDrawEllipse Record 143
EmfPlusDrawImage packet 144
EmfPlusDrawImage Record 144
EmfPlusDrawImagePoints packet 146
EmfPlusDrawImagePoints Record 146
EmfPlusDrawLines packet 148
EmfPlusDrawLines Record 148
EmfPlusDrawPath example 232
EmfPlusDrawPath packet 150
EmfPlusDrawPath Record 150
EmfPlusDrawPie packet 151
EmfPlusDrawPie Record 151
EmfPlusDrawRects packet 152
EmfPlusDrawRects Record 152
EmfPlusDrawString packet 153
EmfPlusDrawString Record 153
EmfPlusEndContainer example 249
EmfPlusEndContainer packet 179
EmfPlusEndContainer Record 179
EmfPlusEndOfFile packet 128
EmfPlusEndOfFile Record 128
EmfPlusFillClosedCurve packet 155
EmfPlusFillClosedCurve Record 155
EmfPlusFillEllipse packet 158
EmfPlusFillEllipse Record 158
EmfPlusFillPath Object 89
EmfPlusFillPath packet 159
EmfPlusFillPath Record 159
EmfPlusFillPath_Object packet 89
EmfPlusFillPie packet 160
EmfPlusFillPie Record 160
EmfPlusFillPolygon packet 162
EmfPlusFillPolygon Record 162
EmfPlusFillRects packet 164
EmfPlusFillRects Record 164
EmfPlusFillRegion packet 165
EmfPlusFillRegion Record 165
EmfPlusFocusScaleData Object 89
EmfPlusFocusScaleData packet 89
EmfPlusFont Object 69
EmfPlusFont packet 69
EmfPlusGetDC packet 128
EmfPlusGetDC Record 128
EmfPlusGraphicsVersion Object 90
EmfPlusGraphicsVersion packet 90
EmfPlusHatchBrushData Object 90
EmfPlusHatchBrushData packet 90
EmfPlusHeader example 200
EmfPlusHeader packet 129
EmfPlusHeader Record 129
EmfPlusImage Object 70
EmfPlusImage packet 70
EmfPlusImageAttributes Object 70
EmfPlusImageAttributes packet 70
EmfPlusInteger15 Object 91
EmfPlusInteger15 packet 91
EmfPlusInteger7 Object 91
EmfPlusInteger7 packet 91
EmfPlusLanguageIdentifier Object 92
EmfPlusLanguageIdentifier packet 92
EmfPlusLinearGradientBrushData Object 92
EmfPlusLinearGradientBrushData packet 92
EmfPlusLinearGradientBrushOptionalData Object 93
EmfPlusLinearGradientBrushOptionalData packet 93
EmfPlusLinePath Object 94
EmfPlusLinePath packet 94
EmfPlusMetafile Object 95
EmfPlusMetafile packet 95
EmfPlusMultiplyWorldTransform example (section 3.2.32.6 218, section 3.2.32.8 220)
EmfPlusMultiplyWorldTransform packet 187
EmfPlusMultiplyWorldTransform Record 187
EmfPlusObject example (section 3.2.32.21 227, section 3.2.32.22 229, section 3.2.33.1 230)
EmfPlusObject packet 167
EmfPlusObject Record 167
EmfPlusOffsetClip packet 122
EmfPlusOffsetClip Record 122
EmfPlusPalette Object 95
EmfPlusPalette packet 95
EmfPlusPath Object 71
EmfPlusPath packet 71
EmfPlusPathGradientBrushData Object 96
EmfPlusPathGradientBrushData packet 96
EmfPlusPathGradientBrushOptionalData Object 98
EmfPlusPathGradientBrushOptionalData packet 98
EmfPlusPathPointType Object 99
EmfPlusPathPointType packet 99
EmfPlusPathPointTypeRLE Object 99
EmfPlusPathPointTypeRLE packet 99
EmfPlusPen Object 72
EmfPlusPen packet 72
EmfPlusPenData Object 99
EmfPlusPenData packet 99
EmfPlusPenOptionalData Object 100
EmfPlusPenOptionalData packet 100
EmfPlusPoint Object 102
EmfPlusPoint packet 102
EmfPlusPointF Object 103
EmfPlusPointF packet 103
EmfPlusPointR Object 103
EmfPlusPointR packet 103
EmfPlusRect Object 103
EmfPlusRect packet 103
EmfPlusRectF packet 104
EmfPlusRegion Object 73
EmfPlusRegion packet 73
EmfPlusRegionNode Object (section 2.2.2.39 104, section 2.2.2.40 104)
EmfPlusRegionNode packet 104
EmfPlusRegionNodeChildNodes Object 105
EmfPlusRegionNodeChildNodes packet 105
EmfPlusRegionNodePath_Object packet 105
EmfPlusResetClip packet 123
EmfPlusResetClip Record 123
EmfPlusResetWorldTransform packet 187
EmfPlusResetWorldTransform Record 187
EmfPlusRestore example 249
EmfPlusRestore packet 180
EmfPlusRestore Record 180
EmfPlusRotateWorldTransform packet 188
EmfPlusRotateWorldTransform Record 188
EmfPlusSave example 219
EmfPlusSave packet 181

EmfPlusSave Record 181
EmfPlusScaleWorldTransform packet 189
EmfPlusScaleWorldTransform Record 189
EmfPlusSerializableObject packet 168
EmfPlusSerializableObject Record 168
EmfPlusSetAntiAliasMode example (section 3.2.32.1 216, section 3.2.32.11 222)
EmfPlusSetAntiAliasMode packet 171
EmfPlusSetAntiAliasMode Record 171
EmfPlusSetClipPath packet 123
EmfPlusSetClipPath Record 123
EmfPlusSetClipRect packet 124
EmfPlusSetClipRect Record 124
EmfPlusSetClipRegion packet 125
EmfPlusSetClipRegion Record 125
EmfPlusSetCompositingMode packet 171
EmfPlusSetCompositingMode Record 171
EmfPlusSetCompositingQuality example (section 3.2.32.2 216, section 3.2.32.12 222)
EmfPlusSetCompositingQuality packet 172
EmfPlusSetCompositingQuality Record 172
EmfPlusSetInterpolationMode example (section 3.2.32.3 217, section 3.2.32.13 223)
EmfPlusSetInterpolationMode packet 173
EmfPlusSetInterpolationMode Record 173
EmfPlusSetPageTransform example 224
EmfPlusSetPageTransform packet 190
EmfPlusSetPageTransform Record 190
EmfPlusSetPixelOffsetMode example (section 3.2.32.4 217, section 3.2.32.14 223)
EmfPlusSetPixelOffsetMode packet 173
EmfPlusSetPixelOffsetMode Record 173
EmfPlusSetRenderingOrigin packet 174
EmfPlusSetRenderingOrigin Record 174
EmfPlusSetTextContrast packet 175
EmfPlusSetTextContrast Record 175
EmfPlusSetTextRenderingHint example (section 3.2.32.5 218, section 3.2.32.15 223)
EmfPlusSetTextRenderingHint packet 176
EmfPlusSetTextRenderingHint Record 176
EmfPlusSetTSClip packet 182
EmfPlusSetTSClip Record 182
EmfPlusSetTSGraphics packet 184
EmfPlusSetTSGraphics Record 184
EmfPlusSetWorldTransform example (section 3.2.32.9 220, section 3.2.32.17 224, section 3.2.32.18 225, section 3.2.32.19 226, section 3.2.32.20 227, section 3.2.67.3 250)
EmfPlusSetWorldTransform packet 191
EmfPlusSetWorldTransform Record 191
EmfPlusSolidBrushData Object 106
EmfPlusSolidBrushData packet 106
EmfPlusStringFormat Object 74
EmfPlusStringFormat packet 74
EmfPlusStringFormatData Object 106
EmfPlusStringFormatData packet 106
EmfPlusTextureBrushData Object 107
EmfPlusTextureBrushData packet 107
EmfPlusTextureBrushOptionalData Object 107
EmfPlusTextureBrushOptionalData packet 107
EmfPlusTransformMatrix Object 108
EmfPlusTransformMatrix packet 108
EmfPlusTranslateWorldTransform packet 191
EmfPlusTranslateWorldTransform Record 191
EMR_BITBLT example 251
EMR_COMMENT_BEGINGROUP example 243
EMR_COMMENT_EMFPLUS example (section 3.2.2 199, section 3.2.32 215, section 3.2.67 248, section 3.2.70 253)
EMR_COMMENT_ENDGROUP example 247
EMR_DELETEOBJECT example 215
EMR_EOF example 254
EMR_EXTCREATEFONTINDIRECTW example 212
EMR_MODIFYWORLDTRANSFORM example (section 3.2.25 210, section 3.2.56 242, section 3.2.60 244, section 3.2.61 245, section 3.2.62 246)
EMR_MOVETOEX example (section 3.2.23 209, section 3.2.54 241)
EMR_RESTOREDC example (section 3.2.34 232, section 3.2.35 233, section 3.2.37 233, section 3.2.66 248, section 3.2.69 252)
EMR_ROUNDRECT example 247
EMR_SAVEDC example (section 3.2.3 200, section 3.2.5 201, section 3.2.7 202, section 3.2.38 234)
EMR_SELECTOBJECT example (section 3.2.10 203, section 3.2.11 204, section 3.2.12 204, section 3.2.29 214, section 3.2.30 215, section 3.2.36 233, section 3.2.41 235, section 3.2.42 235, section 3.2.43 236, section 3.2.63 246)
EMR_SELECTPALETTE example (section 3.2.13 204, section 3.2.44 236)
EMR_SETBKCOLOR example (section 3.2.14 205, section 3.2.45 237)
EMR_SETBKMODE example (section 3.2.16 206, section 3.2.47 237)
EMR_SETBRUSHORGEX example (section 3.2.21 208, section 3.2.27 211, section 3.2.52 240)
EMR_SETICMMODE example (section 3.2.4 201, section 3.2.6 202)
EMR_SETLAYOUT example (section 3.2.8 202, section 3.2.26 211, section 3.2.39 234, section 3.2.57 242)
EMR_SETMETARGN example (section 3.2.9 203, section 3.2.40 235)
EMR_SETMITERLIMIT example (section 3.2.22 208, section 3.2.53 240)
EMR_SETPOLYFILLMODE example (section 3.2.17 206, section 3.2.48 238)
EMR_SETROP2 example (section 3.2.18 207, section 3.2.49 238)
EMR_SETSTRETCHBLTMODE example (section 3.2.19 207, section 3.2.50 239)
EMR_SETTEXTALIGN example (section 3.2.20 207, section 3.2.51 239)
EMR_SETTEXTCOLOR example (section 3.2.15 205, section 3.2.46 237)
EMR_SETWORLDTRANSFORM example (section 3.2.24 209, section 3.2.55 241, section 3.2.59 244)
EMRCOMMENT_EMFPLUS example 229
Enumeration constant types 23
Examples 193
  byte ordering example 193
  EMF+ Metafile Example 194
  EMF+ metafile examples 194
  EMF+ String Drawing Example 254
  EmfPlusBeginContainerNoParams example 221
  EmfPlusDrawPath example 232
  EmfPlusEndContainer example 249

EmfPlusHeader example 200
EmfPlusMultiplyWorldTransform example (section 3.2.32.6 218, section 3.2.32.8 220)
EmfPlusObject example (section 3.2.32.21 227, section 3.2.32.22 229, section 3.2.33.1 230)
EmfPlusRestore example 249
EmfPlusSave example 219
EmfPlusSetAntiAliasMode example (section 3.2.32.1 216, section 3.2.32.11 222)
EmfPlusSetCompositingQuality example (section 3.2.32.2 216, section 3.2.32.12 222)
EmfPlusSetInterpolationMode example (section 3.2.32.3 217, section 3.2.32.13 223)
EmfPlusSetPageTransform example 224
EmfPlusSetPixelOffsetMode example (section 3.2.32.4 217, section 3.2.32.14 223)
EmfPlusSetTextRenderingHint example (section 3.2.32.5 218, section 3.2.32.15 223)
EmfPlusSetWorldTransform example (section 3.2.32.9 220, section 3.2.32.17 224, section 3.2.32.18 225, section 3.2.32.19 226, section 3.2.32.20 227, section 3.2.67.3 250)
EMR_BITBLT example 251
EMR_COMMENT_BEGINGROUP example 243
EMR_COMMENT_EMFPLUS example (section 3.2.2 199, section 3.2.32 215, section 3.2.33 229, section 3.2.67 248, section 3.2.70 253)
EMR_COMMENT_ENDGROUP example 247
EMR_DELETEOBJECT example 215
EMR_EOF example 254
EMR_EXTCREATEFONTINDIRECTW example 212
EMR_MODIFYWORLDTRANSFORM example (section 3.2.25 210, section 3.2.56 242, section 3.2.60 244, section 3.2.61 245, section 3.2.62 246)
EMR_MOVETOEX example (section 3.2.23 209, section 3.2.54 241)
EMR_RESTOREDC example (section 3.2.34 232, section 3.2.35 233, section 3.2.37 233, section 3.2.66 248, section 3.2.69 252)
EMR_ROUNDRECT example 247
EMR_SAVEDC example (section 3.2.3 200, section 3.2.5 201, section 3.2.7 202, section 3.2.38 234)
EMR_SELECTOBJECT example (section 3.2.10 203, section 3.2.11 204, section 3.2.12 204, section 3.2.29 214, section 3.2.30 215, section 3.2.36 233, section 3.2.41 235, section 3.2.42 235, section 3.2.43 236, section 3.2.63 246)
EMR_SELECTPALETTE example (section 3.2.13 204, section 3.2.44 236)
EMR_SETBKCOLOR example (section 3.2.14 205, section 3.2.45 237)
EMR_SETBKMODE example (section 3.2.16 206, section 3.2.47 237)
EMR_SETBRUSHORGEX example (section 3.2.21 208, section 3.2.27 211, section 3.2.52 240)
EMR_SETICMMODE example (section 3.2.4 201, section 3.2.6 202)
EMR_SETLAYOUT example (section 3.2.8 202, section 3.2.26 211, section 3.2.39 234, section 3.2.57 242)
EMR_SETMETARGN example (section 3.2.9 203, section 3.2.40 235)
EMR_SETMITERLIMIT example (section 3.2.22 208, section 3.2.53 240)
EMR_SETPOLYFILLMODE example (section 3.2.17 206, section 3.2.48 238)
EMR_SETROP2 example (section 3.2.18 207, section 3.2.49 238)
EMR_SETSTRETCHBLTMODE example (section 3.2.19 207, section 3.2.50 239)
EMR_SETTEXTALIGN example (section 3.2.20 207, section 3.2.51 239)
EMR_SETTEXTCOLOR example (section 3.2.15 205, section 3.2.46 237)
EMR_SETWORLDTRANSFORM example (section 3.2.24 209, section 3.2.55 241, section 3.2.59 244)
managing objects example 193
metafile design examples 193
overview 193

F

Fields - vendor-extensible 22
FilterType enumeration 33
FontStyleBold 62
FontStyleItalic 62
FontStyleStrikeout 62
FontStyleUnderline 62

G

Glossary 10
GraphicsVersion enumeration 33

H

HatchStyle enumeration 34
HotkeyPrefix enumeration 37
HueSaturationLightnessEffect Object 117
HueSaturationLightnessEffect packet 117
HueSaturationLightnessEffectGuid 66

I

ImageDataType enumeration 38
Informative references 18
InterpolationMode enumeration 38
Introduction 10

L

LanguageIdentifiers enumeration 39
LevelsEffect Object 118
LevelsEffect packet 118
LevelsEffectGuid 66
LineCapType enumeration 50
LineJoinType enumeration 51
LineStyle enumeration 51
Localization 21

M

Managing objects example 193
Metafile design examples 193
Metafile structure 18
MetafileDataType enumeration 52

N

Normative references 17

O

Object_Record_Types packet 166
Objects
  EMF+ 67
  overview 23
ObjectType enumeration 52
Overview (synopsis) 18

P

PaletteStyleGrayScale 63
PaletteStyleHalftone 63
PaletteStyleHasAlpha 63
PathPointType enumeration 53
PathPointTypeCloseSubpath 63
PathPointTypeDashMode 63
PathPointTypePathMarker 63
PenAlignment enumeration 53
PenDataCompoundLine 64
PenDataCustomEndCap 64
PenDataCustomStartCap 64
PenDataDashedLine 64
PenDataDashedLineCap 64
PenDataDashedLineOffset 64
PenDataEndCap 64
PenDataJoin 64
PenDataLineStyle 64
PenDataMiterLimit 64
PenDataNonCenter 64
PenDataStartCap 64
PenDataTransform 64
PixelFormat enumeration 54
PixelOffsetMode enumeration 55
Product behavior 257
Property_Record_Types packet 169

R

Record types 121
Records
  EMF+ 121
  types 121
RecordType enumeration 25
RedEyeCorrectionEffect Object 119
RedEyeCorrectionEffect packet 119
RedEyeCorrectionEffectGuid 66
References 17
  informative 18
  normative 17
RegionNodeDataType enumeration 56
Relationship to other protocols 21
Relationship to protocols and other structures 21

S

Security 256
SharpenEffect Object 120
SharpenEffect packet 120
SharpenEffectGuid 66
SmoothingMode enumeration 57
Standard identifier constant types 66
State_Record_Types packet 176
StringAlignment enumeration 57
StringDigitSubstitution enumeration 58
StringFormatBypassGDI 64
StringFormatDirectionRightToLeft 64
StringFormatDirectionVertical 64
StringFormatDisplayFormatControl 64
StringFormatLineLimit 64
StringFormatMeasureTrailingSpaces 64
StringFormatNoClip 64
StringFormatNoFitBlackBox 64
StringFormatNoFontFallback 64
StringFormatNoWrap 64
StringTrimming enumeration 58
Structures
  overview 23
Structures - overview 23

T

Terminal_Server_Record_Types packet 181
TextRenderingHint enumeration 59
TintEffect Object 120
TintEffect packet 120
TintEffectGuid 66
Tracking changes 260
Transform_Record_Types packet 185

U

UnitType enumeration 59

V

Vendor-extensible fields 22
Versioning 21

W

WrapMode enumeration 60